Hybrid Grammars for Parsing of Discontinuous Phrase Structures and Non-Projective Dependency Structures

Kilian Gebhardt*
Technische Universität Dresden

Mark-Jan Nederhof**
University of St. Andrews

Heiko Vogler*
Technische Universität Dresden

We explore the concept of hybrid grammars, which formalize and generalize a range of existing frameworks for dealing with discontinuous syntactic structures. Covered are both discontinuous phrase structures and non-projective dependency structures. Technically, hybrid grammars are related to synchronous grammars, where one grammar component generates linear structures and another generates hierarchical structures. By coupling lexical elements of both components together, discontinuous structures result. Several types of hybrid grammars are characterized. We also discuss grammar induction from treebanks. The main advantage over existing frameworks is the ability of hybrid grammars to separate discontinuity of the desired structures from time complexity of parsing. This permits exploration of a large variety of parsing algorithms for discontinuous structures, with different properties. This is confirmed by the reported experimental results, which show a wide variety of running time, accuracy, and frequency of parse failures.

1. Introduction

Much of the theory of parsing assumes syntactic structures that are trees, formalized such that the children of each node are ordered, and the yield of a tree, that is, the leaves read from left to right, is the sentence. In different terms, each node in the hierarchical syntactic structure of a sentence corresponds to a phrase that is a list of adjacent words, without any gaps. Such a structure is easy to represent in terms of bracketed notation, which is used, for instance, in the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993).

* Department of Computer Science, Technische Universität Dresden, D-01062 Dresden, Germany.
E-mail: {kilian.gebhardt, heiko.vogler}@tu-dresden.de.
** School of Computer Science, University of St. Andrews, North Haugh, St. Andrews, KY16 9SX, UK.

Submission received: 19 November 2015; revised version received: 11 November 2016; accepted for publication: 22 December 2016.

doi:10.1162/COLI_a_00291

Describing syntax in terms of such narrowly defined trees seems most appropriate for relatively rigid word-order languages such as English. Nonetheless, the aforementioned Penn Treebank of English contains traces and other elements that encode additional structure next to the pure tree structure as indicated by the brackets. This is in keeping with observations that even English cannot be described adequately without a more general form of trees, allowing for so-called **discontinuity** (McCawley 1982; Stucky 1987). In a discontinuous structure, the set of leaves dominated by a node of the tree need not form a contiguous sequence of words, but may comprise one or more gaps. The need for discontinuous structures tends to be even greater for languages with relatively free word order (Kathol and Pollard 1995; Müller 2004).

In the context of dependency parsing (Kübler, McDonald, and Nivre 2009), the more specific term **non-projectivity** is used instead of, or next to, *discontinuity*. See Rambow (2010) for a discussion of the relation between constituent and dependency structures and see Maier and Lichte (2009) for a comparison of discontinuity and non-projectivity. As shown by, for example, Hockenmaier and Steedman (2007) and Evang and Kallmeyer (2011), discontinuity encoded using traces in the Penn Treebank can be rendered in alternative, and arguably more explicit, forms. In many modern treebanks, discontinuous structures have been given a prominent status (e.g., Böhmová et al. 2000). Figure 1 shows an example of a non-projective dependency structure.

The most established parsing algorithms are compiled out of **context-free grammars** (CFGs), or closely related formalisms such as tree substitution grammars (Sima'an et al. 1994) or regular tree grammars (Brainerd 1969; Gécseg and Steinby 1997). These parsers, which have a time complexity of $\mathcal{O}(n^3)$ for n being the length of the input string, operate by composing adjacent substrings of the input sentence into longer substrings. As a result, the structures they can build directly do not involve any discontinuity. The need for discontinuous syntactic structures thus poses a challenge to traditional parsing algorithms.

One possible solution is commonly referred to as **pseudo-projectivity** in the literature on dependency parsing (Kahane, Nasr, and Rambow 1998; Nivre and Nilsson 2005; McDonald and Pereira 2006). A standard parsing system is trained on a corpus of projective dependency structures that was obtained by applying a **lifting** operation to non-projective structures. In a first pass, this system is applied to unlabeled sentences and produces projective dependencies. In a second pass, the lifting operation is reversed to introduce non-projectivity. A related idea for discontinuous phrase structures is the reversible splitting conversion of Boyd (2007). See also Johnson (2002), Campbell (2004), and Gabbard, Kulick, and Marcus (2006).

The two passes of pseudo-projective dependency parsing need not be strictly separated in time. For example, one way to characterize the algorithm by Nivre (2009) is that it combines the first pass with the second. Here the usual one-way input tape is replaced by a buffer. A non-topmost element from the parsing stack, which holds a word previously read from the input sentence, can be transferred back to the buffer, and

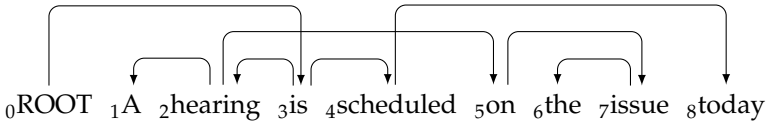

Figure 1
A non-projective dependency structure.

thereby input positions can be effectively swapped. This then results in a non-projective dependency structure.

A second potential solution to obtain syntactic structures that go beyond context-free power is to use more expressive grammatical formalisms. One approach proposed by Reape (1989, 1994) is to separate linear order from the parent–child relation in syntactic structure, and to allow shuffling of the order of descendants of a node, which need not be its direct children. The set of possible orders is restricted by linear precedence constraints. A further restriction may be imposed by compaction (Kathol and Pollard 1995). As discussed by Fouvry and Meurers (2000) and Daniels and Meurers (2002), this may lead to exponential parsing complexity; see also Daniels and Meurers (2004). Separating linear order from the parent–child relation is in the tradition of head-driven phrase structure grammar (HPSG), where grammars are commonly hand-written. This differs from our objectives to induce grammars automatically from training data, as will become clear in the following sections.

To stay within a polynomial time complexity, one may also consider **tree adjoining grammars** (TAGs), which can describe strictly larger classes of word order phenomena than CFGs (Rambow and Joshi 1997). The resulting parsers have a time complexity of $\mathcal{O}(n^6)$ (Vijay-Shankar and Joshi 1985). However, the derived trees they generate are still continuous. Although their derivation trees may be argued to be discontinuous, these by themselves are not normally the desired syntactic structures. Moreover, it was argued by Becker, Joshi, and Rambow (1991) that further additions to TAGs are needed to obtain adequate descriptions of certain non-context-free phenomena. These additions further increase the time complexity.

In order to obtain desired syntactic structures, one may combine TAG parsing with an idea that is related to that of pseudo-projectivity. For example, Kallmeyer and Kuhlmann (2012) propose a transformation that turns a derivation tree of a (lexicalized) TAG into a non-projective dependency structure. The same idea has been applied to derivation trees of other formalisms, in particular (lexicalized) **linear context-free rewriting systems** (LCFRSs) (Kuhlmann 2013), whose weak generative power subsumes that of TAGs.

Parsers more powerful than those for CFGs often incur high time costs. In particular, LCFRS parsers have a time complexity that is polynomial in the sentence length, but with a degree that is determined by properties of the grammar. This degree typically increases with the amount of discontinuity in the desired structures. Difficulties in running LCFRS parsers for natural languages are described, for example, by Kallmeyer and Maier (2013).

In the architectures we have discussed, the common elements are:

- a grammar, in some fixed formalism, that determines the set of sentences that are accepted, and
- a procedure to build (discontinuous) structures, guided by the derivation of input sentences.

The purpose of this article is to explore a theoretical framework that allows us to capture a wide range of parsing architectures that all share these two common elements. At the core of this framework lies a formalism called **hybrid grammar**, introduced in Nederhof and Vogler (2014). Such a grammar consists of a string grammar and a tree grammar. Derivations are coupled, as in synchronous grammars (Shieber and Schabes 1990; Satta and Peserico 2005). In addition, each occurrence of a terminal symbol in the string grammar is coupled to an occurrence of a terminal symbol in the tree grammar. The string grammar defines the set of accepted sentences. The tree grammar, whose rules are tied

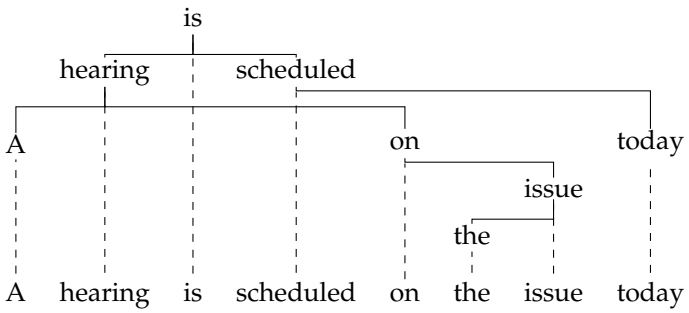

Figure 2
A hybrid tree corresponding to the non-projective dependency structure of Figure 1.

to the rules of the string grammar for synchronous rewriting, determines the resulting syntactic structures, which may be discontinuous. The way the syntactic structures are obtained critically relies on the coupling of terminal symbols in the two component grammars, which is where our theory departs from that of synchronous grammars. A hybrid grammar generates a set of **hybrid trees**;[1] Figure 2 shows an example of a hybrid tree, which corresponds to the non-projective dependency structure of Figure 1.

The general concept of hybrid grammars leaves open the choice of the string grammar formalism and that of the tree grammar formalism. In this article we consider simple macro grammars (Fischer 1968) and LCFRSs as string grammar formalisms. The tree grammar formalisms we consider are simple context-free tree grammars (Rounds 1970) and **simple definite clause programs** (sDCP), inspired by Deransart and Małuszynski (1985). This gives four combinations, each leading to one class of hybrid grammars. In addition, more fine-grained subclasses can be defined by placing further syntactic restrictions on the string and tree formalisms.

To place hybrid grammars in the context of existing parsing architectures, let us consider classical grammar induction from a treebank, for example, for context-free grammars (Charniak 1996) or for LCFRSs (Maier and Søgaard 2008; Kuhlmann and Satta 2009). Rules are extracted directly from the trees in the training set, and unseen input strings are consequently parsed according to these structures. Grammars induced in this way can be seen as restricted hybrid grammars, in which no freedom exists in the relation between the string component and the tree component. In particular, the presence of discontinuous structures generally leads to high time complexity of string parsing. In contrast, the framework in this article detaches the string component of the grammar from the tree component. Thereby the parsing process of input strings is no longer bound to follow the tree structures, while the same tree structures as before can still be produced, provided the tree component is suitably chosen. This allows string parsing with low time complexity in combination with production of discontinuous trees.

Nederhof and Vogler (2014) presented experiments with various subclasses of hybrid grammars for the purpose of constituent parsing. Trade-offs between speed and accuracy

1 The term "hybrid tree" was used before by Lu et al. (2008), also for a mixture of a tree structure and a linear structure, generated by a probabilistic model. However, the linear "surface" structure was obtained by a simple left-to-right tree traversal, whereas a meaning representation was obtained by a slightly more flexible traversal of the same tree. The emphasis in the current article is rather on separating the linear structure from the tree structure. Note that similar distinctions of multiple **strata** were made before in both constituent linguistics (see, e.g., Chomsky 1981) and dependency linguistics (see, e.g., Mel'čuk 1988).

were identified. In the present article, we extend our investigation to dependency parsing. This includes induction of a hybrid grammar from a dependency treebank. Before turning to the experiments, we present several completeness results about existence of hybrid grammars generating non-projective dependency structures.

This article is organized as follows. After preliminaries in Section 2, Section 3 defines hybrid trees. These are able to capture both discontinuous phrase structures and non-projective dependency structures. Thanks to the concept of hybrid trees, there will be no need, later in the article, to distinguish between hybrid grammars for constituent parsing and hybrid grammars for dependency parsing. To make this article self-contained, we define two existing string grammar formalisms and two tree grammar formalisms in Section 4. The four classes of hybrid grammars that result by combining these formalisms are presented in Section 5, which also discusses how to use them for parsing.

How to induce hybrid grammars from treebanks is discussed in Section 6. Section 7 reports on experiments that provide proof of concept. In particular, LCFRS/sDCP-hybrid grammars are induced from corpora of dependency structures and phrase structures and employed to predict the syntactic structure of unlabeled sentences. It is demonstrated that hybrid grammars allow a wide variety of results, in terms of time complexity, accuracy, and frequency of parse failures. How hybrid grammars relate to existing ideas is discussed in Section 8.

The text refers to a number of theoretical results that are not central to the main content of this article. In order to preserve the continuity of the discussion, we have deferred their proofs to appendices.

2. Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. For each $n \in \mathbb{N}_+$, we let $[n]$ stand for the set $\{1, \ldots, n\}$, and we let $[0]$ stand for $\emptyset$. We write $[n]_0$ to denote $[n] \cup \{0\}$. We fix an infinite list $x_1, x_2, \ldots$ of pairwise distinct **variables**. We let $X = \{x_1, x_2, x_3, \ldots\}$ and $X_k = \{x_1, \ldots, x_k\}$ for each $k \in \mathbb{N}$. For any set A, the power set of A is denoted by $\mathcal{P}(A)$.

A **ranked set** Δ is a set of symbols associated with a rank function assigning a number $\mathrm{rk}_\Delta(\delta) \in \mathbb{N}$ to each symbol $\delta \in \Delta$. A **ranked alphabet** is a ranked set with a finite number of symbols. We let $\Delta^{(k)}$ denote $\{\delta \in \Delta \mid \mathrm{rk}_\Delta(\delta) = k\}$.

The following definitions were inspired by Seki and Kato (2008). The sets of **terms** and **sequence-terms** (**s-terms**) over ranked set Δ, with variables in some set $Y \subseteq X$, are denoted by $T_\Delta(Y)$ and $T^*_\Delta(Y)$, respectively, and defined inductively as follows:

(i) $Y \subseteq T_\Delta(Y)$,

(ii) if $k \in \mathbb{N}$, $\delta \in \Delta^{(k)}$ and $s_i \in T^*_\Delta(Y)$ for each $i \in [k]$, then $\delta(s_1, \ldots, s_k) \in T_\Delta(Y)$, and

(iii) if $n \in \mathbb{N}$ and $t_i \in T_\Delta(Y)$ for each $i \in [n]$, then $\langle t_1, \ldots, t_n \rangle \in T^*_\Delta(Y)$.

We let T^*_Δ and T_Δ stand for $T^*_\Delta(\emptyset)$ and $T_\Delta(\emptyset)$, respectively. Throughout this article, we use variables such as s and s_i for s-terms and variables such as t and t_i for terms. The **length** $|s|$ of a s-term $s = \langle t_1, \ldots, t_n \rangle$ is n.

The justification for using s-terms as defined here is that they provide the required flexibility for dealing with both strings and unranked trees, in combination with derivational nonterminals in various kinds of grammar. By using an alphabet $\Delta = \Delta^{(0)}$ one can represent strings. For instance, if Δ contains the symbols a and b, then the

s-term $\langle a(), b(), a()\rangle$ denotes the string *a b a*. We will therefore refer to such s-terms simply as strings.

By using an alphabet $\Delta = \Delta^{(1)}$ one may represent trees without fixing the number of child nodes that a node with a certain label should have. Conceptually, one may think of such node labels as unranked, as is common in parsing theory of natural language. For instance, the s-term $\langle a(\langle b(\langle\rangle), a(\langle\rangle)\rangle)\rangle$ denotes the unranked tree $a(b, a)$. We will therefore refer to such s-terms simply as trees, and we will sometimes use familiar terminology, such as "node," "parent," and "sibling," as well as common graphical representations of trees.

If theoretical frameworks require trees over ranked alphabets in the conventional sense (without s-terms), one may introduce a distinguished symbol *cons* of rank 2, replacing each s-term of length greater than 1 by an arrangement of subterms combined using occurrences of that symbol. Another symbol *nil* of rank 0 may be introduced to replace each s-term of length 0. Hence $\delta(\langle\alpha(\langle\rangle), \beta(\langle\rangle), \gamma(\langle\rangle)\rangle)$ could be more conventionally written as $\delta(cons(\alpha(nil), cons(\beta(nil), \gamma(nil))))$.

Concatenation of s-terms is given by $\langle t_1, \ldots, t_n\rangle \cdot \langle t_{n+1}, \ldots, t_{n+m}\rangle = \langle t_1, \ldots, t_{n+m}\rangle$. Sequences such as $s_1, \ldots, s_k$ or $x_1, \ldots, x_k$ will typically be abbreviated to $s_{1,k}$ or $x_{1,k}$, respectively. For $\delta \in \Delta^{(0)}$ we sometimes abbreviate $\delta()$ to δ.

In examples we also abbreviate $\langle t_1, \ldots, t_n\rangle$ to $t_1 \cdots t_n$—that is, omitting the angle brackets and commas. In particular, for $n = 0$, the s-term $\langle\rangle$ is abbreviated by ε. Moreover, we sometimes abbreviate $\delta(\langle\rangle)$ to δ. Whether δ then stands for $\delta(\langle\rangle)$ or for $\delta()$ depends on whether $\delta \in \Delta^{(1)}$ or $\delta \in \Delta^{(0)}$, which will be clear from the context.

Subterms in terms or s-terms are identified by **positions** (cf. Gorn addresses). The set of all positions in term t or in s-term s is denoted by $\mathrm{pos}(t)$ or $\mathrm{pos}(s)$, respectively, and defined inductively by:

$$\begin{aligned} \mathrm{pos}(x) &= \{\varepsilon\} \\ \mathrm{pos}(\delta(s_{1,k})) &= \{\varepsilon\} \cup \{ip \mid i \in [k], p \in \mathrm{pos}(s_i)\} \\ \mathrm{pos}(\langle t_{1,n}\rangle) &= \{ip \mid i \in [n], p \in \mathrm{pos}(t_i)\} \end{aligned}$$

Note that a (sub-)s-term does not have a position. We let $\leq_\ell$ denote the lexicographical order of positions. We say that position p is a **parent** of position p' if p' is of the form pij, for some numbers i and j, and we denote $\mathrm{parent}(pij) = p$. For a position p' of length 0 or 1, we set $\mathrm{parent}(p') = \mathrm{nil}$. Conversely, the set of **children** of a position p, denoted by $\mathrm{children}(p)$, contains each position p' with $\mathrm{parent}(p') = p$. The **right sibling** of the position pij, denoted by $\mathrm{right\text{-}sibling}(pij)$, is $pi(j+1)$.

The subterm at position p in a term t (or a s-term s) is defined as follows. For any term t we have $t|_\varepsilon = t$. For a term $t = \delta(s_{1,k})$, $i \in [k], p \in \mathrm{pos}(s_i)$ we have $t|_{ip} = s_i|_p$. For a s-term $s = \langle t_{1,n}\rangle$, $i \in [n], p \in \mathrm{pos}(t_i)$ we have $s|_{ip} = t_i|_p$.

The **label** at position p in a term t is denoted by $t(p)$. In other words, if $t|_p$ equals $\delta(s_1, \ldots, s_k)$ or $x \in X$, then $t(p)$ equals δ or x, respectively. Let $\Gamma \subseteq \Delta$. The subset of $\mathrm{pos}(t)$ consisting of all positions where the label is in Γ is denoted by $\mathrm{pos}_\Gamma(t)$, or formally $\mathrm{pos}_\Gamma(t) = \{p \in \mathrm{pos}(t) \mid t(p) \in \Gamma\}$. Analogously to $t(p)$ and $\mathrm{pos}_\Gamma(t)$ for terms t, one may define $s(p)$ and $\mathrm{pos}_\Gamma(s)$ for s-terms s.

The expression $t[s']_p$ (or $s[s']_p$) denotes the s-term obtained from t (or from s) by replacing the subterm at position p by s-term s'. For any term t we have $t[s']_\varepsilon = s'$. For a term $t = \delta(s_{1,k})$, $i \in [k], p \in \mathrm{pos}(s_i)$ we have $t[s']_{ip} = \langle\delta(s_{1,i-1}, s_i[s']_p, s_{i+1,k})\rangle$. For a s-term $s = \langle t_{1,n}\rangle$, $i \in [n], p \in \mathrm{pos}(t_i)$ we have $s[s']_{ip} = \langle t_{1,i-1}\rangle \cdot t_i[s']_p \cdot \langle t_{i+1,n}\rangle$.

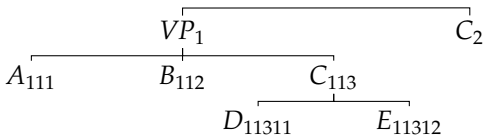

Figure 3
Illustration of the s-term $\langle VP(\langle A(\langle\rangle), B(\langle\rangle), C(\langle D(\langle\rangle), E(\langle\rangle)\rangle)\rangle), C(\langle\rangle)\rangle$ where positions are annotated in subscript at each node.

For term t (or s-term s) with variables in X_k and s-terms s_i ($i \in [k]$), the **first-order substitution** $t[s_{1,k}]$ (or $s[s_{1,k}]$, respectively) denotes the s-term obtained from t (or from s) by replacing each occurrence of any variable x_i by s_i, or formally:

- $x_i[s_{1,k}] = s_i$,
- $\delta(s'_{1,n})[s_{1,k}] = \langle \delta(s'_1[s_{1,k}], \ldots, s'_n[s_{1,k}])\rangle$,
- $\langle t_{1,n}\rangle[s_{1,k}] = t_1[s_{1,k}] \cdot \ldots \cdot t_n[s_{1,k}]$.

If $s \in T^*_\Delta$, $s|_p = \delta(s_{1,k})$ and $s' \in T^*_\Delta(X_k)$, then the **second-order substitution** $s[\![s']\!]_p$ denotes the s-term obtained from s by replacing the subterm at position p by s', with the variables in s' replaced by the corresponding s-terms found immediately below p, or formally $s[\![s']\!]_p = s[s'[s_{1,k}]]_p$.

Example 1
Consider $\Delta = \{A, B, C, \alpha, \beta, \gamma_1, \gamma_2, \delta_1, \delta_2\}$ where $\mathrm{rk}_\Delta(A) = 0$, $\mathrm{rk}_\Delta(B) = \mathrm{rk}_\Delta(C) = 2$, and all other symbols have rank 1. An example of a s-term in $T^*_\Delta(X_2)$ is $s = \alpha(A())\ x_2\ B(x_1, \beta)$, which is short for:

$$\langle \alpha(\langle A()\rangle), x_2, B(\langle x_1\rangle, \langle \beta(\langle\rangle)\rangle)\rangle$$

We have $|s| = 3$, and $\mathrm{pos}(s) = \{1, 111, 2, 3, 311, 321\}$. Note that 31 is not a position, as positions must point to terms, not s-terms. Further $s|_1 = \alpha(A())$, $s|_{111} = A()$, $s|_{311} = x_1$, $s(2) = x_2$, and $s(3) = B$.

An example of first-order substitution is $s[\delta_1, \delta_2] = \alpha(A())\ \delta_2\ B(\delta_1, \beta)$. With $s' = \gamma_1\ C(\delta_1, \delta_2)\ \gamma_2$, an example of second-order substitution is:

$$s'[\![s]\!]_2 = \gamma_1\ \alpha(A())\ \delta_2\ B(\delta_1, \beta)\ \gamma_2$$

A s-term over a ranked alphabet $\Delta = \Delta^{(1)}$ and its positions are illustrated in Figure 3. ■

3. Hybrid Trees

The purpose of this section is to unify existing notions of non-projective dependency structures and discontinuous phrase structures, formalized using s-terms.

We fix a ranked alphabet $\Sigma = \Sigma^{(1)}$ and a subset $\Gamma \subseteq \Sigma$. A **hybrid tree** over (Γ, Σ) is a pair $h = (s, \leq_s)$, where $s \in T^*_\Sigma$ and $\leq_s$ is a total order on $\mathrm{pos}_\Gamma(s)$. In words, a hybrid tree combines hierarchical structure, in the form of a s-term over the full alphabet Σ, with a linear structure, which can be seen as a string over $\Gamma \subseteq \Sigma$. This string will be denoted by $\mathrm{str}(h)$. Formally, let $\mathrm{pos}_\Gamma(s) = \{p_1, \ldots, p_n\}$ with $p_i \leq_s p_{i+1}$ ($i \in [n-1]$). Then

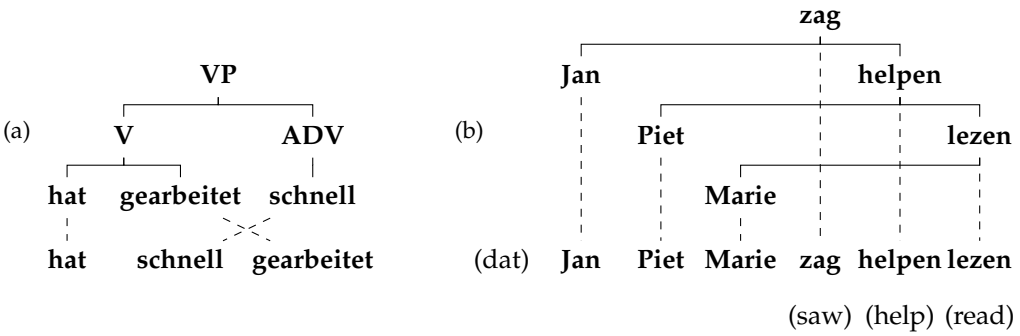

Figure 4
(a) Discontinuous phrase structure for German "[...] hat schnell gearbeitet" ("[...] has worked quickly"), represented as hybrid tree. (b) Non-projective dependency structure for "Jan Piet Marie zag helpen lezen," represented as hybrid tree.

$\mathrm{str}(h) = s(p_1) \cdots s(p_n)$. In order to avoid the treatment of pathological cases we assume that $s \neq \langle\rangle$ and $\mathrm{pos}_\Gamma(s) \neq \emptyset$.

A hybrid tree $(s, \leq_s)$ is a **phrase structure** if $\leq_s$ is a total order on the leaves of s. The elements of Γ would typically represent lexical items, and the elements of $\Sigma \setminus \Gamma$ would typically represent syntactic categories. A hybrid tree $(s, \leq_s)$ is a **dependency structure** if $\Gamma = \Sigma$, whereby the linear structure of a hybrid tree involves all of its nodes, which represent lexical items. Our dependency structures generalize **totally ordered trees** (Kuhlmann and Niehren 2008) by considering s-terms instead of usual terms over a ranked alphabet.

We say that a phrase structure $(s, \leq_s)$ over (Γ, Σ) is **continuous** if for each $p \in \mathrm{pos}(s)$ the set $\mathrm{pos}_\Gamma(s|_p)$ is a complete span, that is, if the following condition holds: if p_1, p_2, p' satisfy $pp_1, p', pp_2 \in \mathrm{pos}_\Gamma(s)$ and $pp_1 \leq_s p' \leq_s pp_2$, then $p' = pp_3$ for some p_3. If the same condition holds for a dependency structure $(s, \leq_s)$, then we say that $(s, \leq_s)$ is **projective**. If the condition is not satisfied, then we call a phrase structure **discontinuous** and a dependency structure **non-projective**.

Example 2
A simple example of a discontinuous phrase structure following Seifert and Fischer (2004) is $h = (s, \leq_s)$, where:

$$s = \langle \mathbf{VP}(\langle \mathbf{V}(\langle \mathbf{hat}, \mathbf{gearbeitet} \rangle), \mathbf{ADV}(\langle \mathbf{schnell} \rangle) \rangle) \rangle$$

and $\leq_s$ is given by $11111 \leq_s 11112 \leq_s 11211$. It is graphically represented in Figure 4a. The bottom line indicates the word order in German, with adverb **schnell** [quickly] separating the two verbs of the verb phrase. The dashed lines connect the leaves of the tree structure to the total order. (Alternative analyses exist that do not require discontinuity; we make no claim that the shown structure is the most linguistically adequate.) ■

Example 3
The phenomenon of cross-serial dependencies in Dutch (Bresnan et al. 1982) is illustrated in Figure 4b, using a non-projective dependency structure. ■

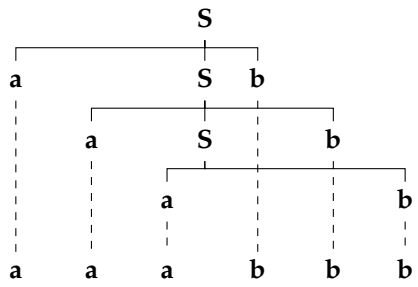

Figure 5
Abstract representation of cross-serial dependencies in Dutch as discontinuous phrase structure.

Example 4
Figure 5 gives an abstract rendering of cross-serial dependencies in Dutch, this time in terms of discontinuous phrase structure. ■

4. Basic Grammatical Formalisms

The concept of hybrid grammars is illustrated in Section 5, first on the basis of a coupling of linear context-free rewriting systems and simple definite clause programs, and then three more such couplings are introduced that further involve simple macro grammars and simple context-free tree grammars. In the current section we discuss these basic classes of grammars, starting with the simplest ones.

4.1 Macro Grammars

The definitions in this section are very close to those in Fischer (1968) with the difference that the notational framework of s-terms is used for strings, as in Seki and Kato (2008).

A **macro grammar** (MG) is a tuple $G = (N, S, \Gamma, P)$, where N is a ranked alphabet of **nonterminals**, $S \in N^{(0)}$ is the **start symbol**, $\Gamma = \Gamma^{(0)}$ is a ranked alphabet of **terminals** and $\Gamma \cap N = \emptyset$, and P is a finite set of **rules**, each of the form:

$$A(x_{1,k}) \to r \tag{1}$$

where $A \in N^{(k)}$ and $r \in T^*_{N \cup \Gamma}(X_k)$. A macro grammar in which each nonterminal has rank 0 is a CFG.

We write $\Rightarrow_G^{p,\rho}$ for the "derives" relation, using rule $\rho = A(x_{1,k}) \to r$ at position p in a s-term. Formally, we write $s_1 \Rightarrow_G^{p,\rho} s_2$ if $s_1 \in T^*_{N\cup\Gamma}$, $s_1(p) = A$ and $s_2 = s_1[\![r]\!]_p$. We write $s_1 \Rightarrow_G s_2$ if $s_1 \Rightarrow_G^{p,\rho} s_2$ for some p and ρ, and $\Rightarrow_G^*$ is the reflexive, transitive closure of $\Rightarrow_G$. Derivation in i steps is denoted by $\Rightarrow_G^i$. The (string) language induced by macro grammar G is $[G] = \{s \in T^*_\Gamma \mid \langle S \rangle \Rightarrow_G^* s\}$.

In the sequel we will focus our attention on macro grammars with the property that for each rule $A(x_{1,k}) \to r$ and each $i \in [k]$, variable x_i has exactly one occurrence in r. In this article, such grammars will be called **simple** macro grammars (sMGs).

Example 5
A sMG and a derivation are the following, once more motivated by cross-serial dependencies:

$$\begin{array}{ll} S \to A(\varepsilon) & S \Rightarrow_G A(\varepsilon) \\ A(x_1) \to \mathbf{a}\, A(x_1\ \mathbf{a}) & \Rightarrow_G \mathbf{a}\, A(\mathbf{a}) \\ A(x_1) \to \mathbf{b}\, A(x_1\ \mathbf{b}) & \Rightarrow_G \mathbf{a}\ \mathbf{b}\, A(\mathbf{a}\ \mathbf{b}) \\ A(x_1) \to x_1 & \Rightarrow_G \mathbf{a}\ \mathbf{b}\ \mathbf{a}\ \mathbf{b} \end{array}$$

All derived strings are of the form ww, where w is a string over $\{\mathbf{a}, \mathbf{b}\}$. In this and in the following examples, bold letters (which may be lower-case or upper-case) represent terminals and upper-case italic letters represent nonterminals. ■

4.2 Context-free Tree Grammars

The definitions in this section are a slight generalization of those in Rounds (1970) and Engelfriet and Schmidt (1977, 1978), as here they involve s-terms. In Section 3 we already argued that the extra power due to s-terms can be modeled using fixed symbols *cons* and *nil* and is therefore not very significant in itself. The benefit of the generalization lies in the combination with macro grammars, as discussed in Section 5.

A (generalized) **context-free tree grammar** (CFTG) is a tuple $G = (N, S, \Sigma, P)$, where Σ is a ranked alphabet with $\Sigma = \Sigma^{(1)}$ and N, S, and P are as for macro grammars except that Γ is replaced by Σ in the specification of the rules.

The "derives" relation $\Rightarrow_G^{p,\rho}$ and other relevant notation are defined as for macro grammars. Note that the language $[G] = \{s \in T_\Sigma^* \mid \langle S \rangle \Rightarrow_G^* s\}$ induced by a CFTG G is not a string language but a tree language, or more precisely, its elements are **sequences** of trees.

As for macro grammars, we will focus our attention on CFTGs with the property that for each rule $A(x_{1,k}) \to r$ and each $i \in [k]$, variable x_i has exactly one occurrence in r. In this article, such grammars will be called **simple** context-free tree grammars (sCFTGs). Note that if $N = N^{(0)}$, then a sCFTGs is a **regular tree grammar** (Brainerd 1969; Gécseg and Steinby 1997). sCFTGs are a natural generalization of the widely used TAGs; see Kepser and Rogers (2011), Maletti and Engelfriet (2012), and Gebhardt and Osterholzer (2015).

Example 6
A sCFTG and a derivation are provided here. This example models a simple recursive structure, where **a** could stand for a noun, **b** for a verb, and **c** for an adverb that modifies exactly one of the verbs.

$$\begin{array}{ll} S \to A(\mathbf{c}) & S \Rightarrow_G A(\mathbf{c}) \\ A(x_1) \to \mathbf{S}(\mathbf{a}\ A(x_1)\ \mathbf{b}) & \Rightarrow_G \mathbf{S}(\mathbf{a}\ A(\mathbf{c})\ \mathbf{b}) \\ A(x_1) \to \mathbf{S}(\mathbf{a}\ A(\varepsilon)\ \mathbf{b}\ x_1) & \Rightarrow_G \mathbf{S}(\mathbf{a}\ \mathbf{S}(\mathbf{a}\ A(\varepsilon)\ \mathbf{b}\ \mathbf{c})\ \mathbf{b}) \\ A(x_1) \to \mathbf{S}(\mathbf{a}\ \mathbf{b}\ x_1) & \Rightarrow_G \mathbf{S}(\mathbf{a}\ \mathbf{S}(\mathbf{a}\ \mathbf{S}(\mathbf{a}\ \mathbf{b})\ \mathbf{b}\ \mathbf{c})\ \mathbf{b}) \end{array}$$ ■

4.3 Linear Context-free Rewriting Systems

In Vijay-Shanker, Weir, and Joshi (1987), the semantics of LCFRS is introduced by distinguishing two phases. In the first phase, a tree over function symbols is generated by

a regular tree grammar. In the second phrase, the function symbols are interpreted, each composing a sequence of tuples of strings into another tuple of strings. This formalism is equivalent to the multiple CFGs of Seki et al. (1991). We choose a notation similar to that of the formalisms discussed before, which will also enable us to couple these string-generating grammars to tree-generating grammars, as will be discussed later.

A linear context-free rewriting system (LCFRS) is a tuple $G = (N, S, \Gamma, P)$, where N is a ranked alphabet of **nonterminals**, $S \in N^{(1)}$ is the **start symbol**, $\Gamma = \Gamma^{(0)}$ is a ranked alphabet of **terminals** and $\Gamma \cap N = \emptyset$, and P is a finite set of **rules**, each of the form:

$$A_0(s_{1,k_0}) \to \langle A_1(x_{1,m_1}), A_2(x_{m_1+1,m_2}), \ldots, A_n(x_{m_{n-1}+1,m_n}) \rangle \tag{2}$$

where $n \in \mathbb{N}$, $A_i \in N^{(k_i)}$ for each $i \in [n]_0$, and $m_i = \sum_{j:1 \le j \le i} k_j$ for each $i \in [n]$, and $s_j \in T^*_\Gamma(X_{m_n})$ for each $j \in [k_0]$. In words, the right-hand side is a s-term consisting of nonterminals A_i ($i \in [n]$), with k_i distinct variables as arguments; there are m_n variables altogether, which is the sum of the ranks of all A_i ($i \in [n]$). The left-hand side is an occurrence of A_0 with each argument being a string of variables and terminals. Furthermore, we demand that each x_j ($j \in [m_n]$) occurs exactly once on the left-hand side. The rank of a nonterminal is called its **fanout** and the largest rank of any nonterminal is called the **fanout** of the grammar.

Given a rule ρ of the form of Equation (2), a **rule instance of** ρ is obtained by choosing some $r_i \in T^*_\Gamma$ for each variable x_i ($i \in [m_n]$), and replacing the two occurrences of x_i in the rule by this r_i. Much as in the preceding sections, $\Rightarrow^\rho_G$ is a binary relation on s-terms, with:

$$s_1 \cdot \langle t \rangle \cdot s_2 \Rightarrow^\rho_G s_1 \cdot r \cdot s_2$$

if $t \to r$ is a rule instance of ρ. We write $\Rightarrow_G$ for $\bigcup_{\rho \in P} \Rightarrow^\rho_G$. The (string) language induced by LCFRS G is $[G] = \{s \in T^*_\Gamma \mid \langle S(s) \rangle \Rightarrow^*_G \langle\rangle\}$. If $\langle S(s) \rangle \Rightarrow^*_G \langle\rangle$, then there are $i \in \mathbb{N}$ and $\rho_1, \ldots, \rho_i$ such that $\langle S(s) \rangle \Rightarrow^{\rho_1}_G \cdots \Rightarrow^{\rho_i}_G \langle\rangle$. We call $\rho_1 \cdots \rho_i$ a **derivation of** G.

A derivation can be represented by a **derivation tree** d (cf. Figure 6), which is obtained by glueing together the rules as they are used in the derivation. The backbone of d is a usual derivation tree of the CFG underlying the LCFRS (nonterminals and solid lines). Each argument of a nonterminal is represented as a box to the right of the nonterminal, and dashed arrows indicate dependencies of values. An x_i above a box

$$\begin{array}{ll} & S(\mathbf{a\ a\ c\ b\ b\ d}) \\ \Rightarrow_G & A(\mathbf{a\ a}, \mathbf{b\ b})\ B(\mathbf{c}, \mathbf{d}) \\ \Rightarrow_G & A(\mathbf{a}, \mathbf{b})\ \ B(\mathbf{c}, \mathbf{d}) \\ \Rightarrow_G & A(\varepsilon, \varepsilon)\ \ B(\mathbf{c}, \mathbf{d}) \\ \Rightarrow_G & \qquad\quad B(\mathbf{c}, \mathbf{d}) \\ \Rightarrow_G & \qquad\quad B(\varepsilon, \varepsilon) \\ \Rightarrow_G & \qquad\quad \varepsilon \end{array}$$

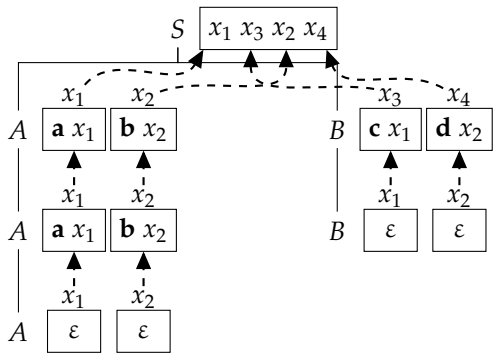

Figure 6
Derivation and derivation tree of the LCFRS in Example 7.

specifies an argument of a nonterminal on the right-hand side of a rule, whereas the content of a box is an argument of the nonterminal on the left-hand side of a rule. The boxes taken as vertices and the dashed arrows taken as edges constitute a **dependency graph**. It can be evaluated by first sorting its vertices topologically and, according to this sorting, substituting in an obvious manner the relevant s-terms into each other. The final s-term that is evaluated in this way is an element of the language $[G]$. This s-term, denoted by $\phi(d)$, is called the **evaluation of** d. The notion of dependency graph originates from attribute grammars (Knuth 1968; Paakki 1995); it should not be confused with the linguistic concept of dependency.

Note that if the rules in a derivation are given, then the choice of r_i for each variable x_i in each rule instance is uniquely determined. For a given string s, the set of all LCFRS derivations (in compact tabular form) can be obtained in polynomial time in the length of s (Seki et al. 1991). See also Kallmeyer and Maier (2010, 2013) for the extension with probabilities.

Example 7
An example of a LCFRS G is the following:

$$S(x_1x_3x_2x_4) \to A(x_1, x_2)\ B(x_3, x_4) \qquad A(\mathbf{a}x_1, \mathbf{b}x_2) \to A(x_1, x_2) \qquad A(\varepsilon, \varepsilon) \to \varepsilon$$
$$B(\mathbf{c}x_1, \mathbf{d}x_2) \to B(x_1, x_2) \qquad B(\varepsilon, \varepsilon) \to \varepsilon$$

A derivation of G and the corresponding derivation tree d is depicted in Figure 6; its evaluation is the s-term $\phi(d) = \mathbf{a\ a\ c\ b\ b\ d}$.

All strings derived by G have the interlaced structure $\mathbf{a}^m\mathbf{c}^n\mathbf{b}^m\mathbf{d}^n$ with $m, n \in \mathbb{N}$, where the i-th occurrence of **a** corresponds to the i-th occurrence of **b** and the i-th occurrence of **c** corresponds to the i-th occurrence of **d**. This resembles cross-serial dependencies in Swiss German (Shieber 1985) in an abstract way; **a** and **c** represent noun phrases with different case markers (dative or accusative) and **b** and **d** are verbs that take different arguments (dative or accusative noun phrases). ■

There is a subclass of LCFRS called **well-nested** LCFRS. Its time complexity of parsing is lower than that of general LCFRS (Gómez-Rodríguez, Kuhlmann, and Satta 2010), and the class of languages it induces is strictly included in the class of languages induced by general LCFRSs (Kanazawa and Salvati 2010). One can see sMGs as syntactic variants of well-nested LCFRSs (cf. footnote 3 of Kanazawa 2009), the former being more convenient for our purposes of constructing hybrid grammars, when the string component is to be explicitly restricted to have the power of sMG / well-nested LCFRS. The class of languages they induce also equals the class of string languages induced by sCFTGs.

4.4 Definite Clause Programs

In this section we describe a particular kind of definite clause program. Our definition is inspired by Deransart and Małuszynski (1985), who investigated the relation between logic programs and attribute grammars, together with the "syntactic single use requirement" from Giegerich (1988). The values produced are s-terms. The induced class of s-term languages is strictly larger than that of sCFTGs (cf. Appendix B).

As discussed subsequently, the class of string languages that results if we take the yields of those s-terms equals the class of string languages induced by LCFRSs. Thereby,

our class of definite clause programs relates to LCFRSs much as the class of sCFTGs relates to sMGs.

A simple definite clause program (sDCP) is a tuple $G = (N, S, \Sigma, P)$, where N is a ranked alphabet of **nonterminals** and $\Sigma = \Sigma^{(1)}$ is a ranked alphabet of **terminals** (as for CFTGs).[2] Moreover, each nonterminal $A \in N$ has a number of arguments, each of which is either an **inherited argument** or a **synthesized argument**. The number of inherited arguments is the **i-rank** and the number of synthesized arguments is the **s-rank** of A; we let $\mathrm{rk}_N(A) = \text{i-rk}(A) + \text{s-rk}(A)$ denote the **rank** of A. The **start symbol** S has only one argument, which is synthesized—that is, $\mathrm{rk}_N(S) = \text{s-rk}(S) = 1$ and $\text{i-rk}(S) = 0$.

A rule is of the form:

$$A_0(x^{(0)}_{1,k_0}, s^{(0)}_{1,k'_0}) \to \langle A_1(s^{(1)}_{1,k'_1}, x^{(1)}_{1,k_1}), \ldots, A_n(s^{(n)}_{1,k'_n}, x^{(n)}_{1,k_n})\rangle \tag{3}$$

where $n \in \mathbb{N}$, $k_0 = \text{i-rk}(A_0)$ and $k'_0 = \text{s-rk}(A_0)$, $k'_i = \text{i-rk}(A_i)$ and $k_i = \text{s-rk}(A_i)$, for $i \in [n]$. The set of variables occurring in the lists $x^{(i)}_{1,k_i}$ ($i \in [n]_0$) equals X_m, where $m = \sum_{i\in[n]_0} k_i$. In other words, every variable from X_m occurs exactly once in all these lists together. This is where values "enter" the rule. Further, the s-terms in $s^{(i)}_{1,k'_i}$ ($i \in [n]_0$) are in $T^*_\Sigma(X_m)$ and together contain each variable in X_m exactly once (syntactic single use requirement). This is where values are combined and "exit" the rule.

The "derives" relation $\Rightarrow_G$ and other relevant notation are defined as for LCFRSs. Thus, in particular, the language induced by sDCP G is $[G] = \{s \in T^*_\Sigma \mid \langle S(s)\rangle \Rightarrow^*_G \langle\rangle\}$, whose elements are now sequences of trees.

In the same way as for LCFRS we can represent a derivation of a sDCP G as derivation tree d. A box in its dependency graph is placed to the left of a nonterminal occurrence if it represents an inherited argument and to the right otherwise. As before, $\phi(d)$ denotes the evaluation of d, which is now a s-term over Σ.

Example 8
A sDCP and a derivation are the following, where the first argument of B is inherited and all other arguments are synthesized:

$$\begin{aligned}
S(x_2) &\to A(x_1)\, B(x_1, x_2) && S(\mathbf{c\ B(c\ B(a\ A()\ b)\ d)\ d}) \\
A(\mathbf{a\ A}(x_1)\ \mathbf{b}) &\to A(x_1) && \Rightarrow\ A\Big(\mathbf{a\ A()\ b}\Big)\, B\Big(\mathbf{a\ A()\ b, c\ B(c\ B(a\ A()\ b)\ d)\ d}\Big) \\
A(\varepsilon) &\to \varepsilon && \Rightarrow^2 A(\varepsilon)\, B\Big(\mathbf{a\ A()\ b,\ c\ B(a\ A()\ b)\ d}\Big) \\
B(x_1, \mathbf{c\ B}(x_2)\ \mathbf{d}) &\to B(x_1, x_2) && \Rightarrow^2 B\Big(\mathbf{a\ A()\ b, a\ A()\ b}\Big) \\
B(x_1, x_1) &\to \varepsilon && \Rightarrow\ \varepsilon
\end{aligned}$$

Figure 7 shows the derivation tree d, whose evaluation is the s-term $\phi(d) =$ **c B(c B(a A() b) d) d** that is derived by the sDCP of this Example. ■

If a sDCP is such that the dependency graph for any derivation contains no cycles, then we say the sDCP contains no cycles. In this case, if the rules in a derivation are given, then the choice of r_i for each variable x_i in each rule instance is uniquely determined, and can be computed in linear time in the size of the derivation. The existence of cycles is

2 The term "simple" will here be used for definite clause programs to have analogous meaning to the term for MGs and CFTGs. This is more restrictive than the term with the same name in Deransart and Małuszynski (1985).

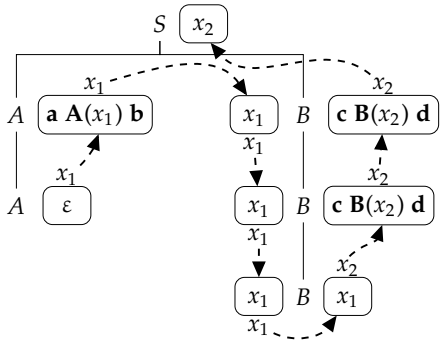

Figure 7
The derivation tree corresponding to the derivation in Example 8.

decidable, as we know from the literature on attribute grammars (Knuth 1968). There are sufficient conditions for absence of cycles, such as the grammar being L-attributed (Bochmann 1976; Deransart, Jourdan, and Lorho 1988). In this article, we will assume that sDCPs contain no cycles. A one-pass computation model can be formalized as a bottom–up tree-generating tree-to-hypergraph transducer (Engelfriet and Vogler 1998). One may alternatively evaluate sDCP arguments using the more general mechanism of unification, as it exists for example in HPSG (Pollard and Sag 1994).

If the sDCP is **single-synthesized**—that is, each nonterminal has exactly one synthesized argument (and any number of inherited arguments)—then there is an equivalent sCFTG. The converse also holds. For proofs, see Appendix A.

Theorem 1
Single-synthesized sDCPs have the same s-term generating power as sCFTGs.

Example 9
The sDCP from Example 8 is transformed into the following sCFTG:

$$\begin{array}{ll} S' \to B'(A') & A' \to \mathbf{a}\ \mathbf{A}(A')\ \mathbf{b} \\ B'(x_1) \to \mathbf{c}\ \mathbf{B}(B'(x_1))\ \mathbf{d} & A' \to \varepsilon \\ B'(x_1) \to x_1 & \end{array}$$

■

A sDCP induces a language of s-terms. By flattening the structure of those s-terms, one obtains a string language. More precisely, we define the flattening function "pre" from $T_\Sigma(X) \cup T^*_\Sigma(X)$ to $T^*_\Sigma(X)$, which returns the sequence of node labels in a pre-order tree traversal. (Alternatively, one could define a function "yield" from $T_\Sigma(X) \cup T^*_\Sigma(X)$ to $T^*_\Gamma(X)$, where Γ is a subset of Σ, which erases symbols in $\Sigma \setminus \Gamma$. The erased symbols would typically be linguistic categories as opposed to lexical elements. This does not affect the validity of what follows, however.) We define recursively:

$$\begin{aligned} \mathrm{pre}(\langle t_1, \ldots, t_n \rangle) &= \mathrm{pre}(t_1) \cdot \ldots \cdot \mathrm{pre}(t_n) \\ \mathrm{pre}(\delta(s)) &= \langle \delta(\langle\rangle) \rangle \cdot \mathrm{pre}(s) \\ \mathrm{pre}(x) &= \langle x \rangle \end{aligned}$$

On the basis of the flattening function, Appendix C proves the following result.

Theorem 2
sDCP have the same string generating power as LCFRS.

5. Hybrid Grammars

A hybrid grammar consists of a string grammar and a tree grammar. Intuitively, the string grammar is used for parsing a given string w, and the tree grammar simultaneously generates a hybrid tree h with $\mathrm{str}(h) = w$. To synchronize these two processes, we couple derivations in the grammars in a way similar to how this is commonly done for synchronous grammars—namely, by **indexed** symbols. However, we apply the mechanism not only to derivational nonterminals but also to terminals.

Let Ω be a ranked alphabet. We define the ranked set $\mathcal{I}(\Omega) = \{\omega^{\boxed{u}} \mid \omega \in \Omega, u \in \mathbb{N}_+\}$, with $\mathrm{rk}_{\mathcal{I}(\Omega)}(\omega^{\boxed{u}}) = \mathrm{rk}_\Omega(\omega)$. Let Δ be another ranked alphabet ($\Omega \cap \Delta = \emptyset$) and $Y \subseteq X$. We let $\mathcal{I}^*_{\Omega,\Delta}(Y)$ be the set of all s-terms $s \in T^*_{\mathcal{I}(\Omega)\cup\Delta}(Y)$ in which each index u occurs at most once.

For a s-term s, let $ind(s)$ be the set of all indices occurring in s. The deindexing function $\mathcal{D}$ removes all indices from a s-term $s \in \mathcal{I}^*_{\Omega,\Delta}(Y)$ to obtain $\mathcal{D}(s) \in T^*_{\Omega\cup\Delta}(Y)$. The set $\mathcal{I}_{\Omega,\Delta}(Y) \subseteq T_{\mathcal{I}(\Omega)\cup\Delta}(Y)$ of terms with indexed symbols is defined much as above. We let $\mathcal{I}^*_{\Omega,\Delta} = \mathcal{I}^*_{\Omega,\Delta}(\emptyset)$ and $\mathcal{I}_{\Omega,\Delta} = \mathcal{I}_{\Omega,\Delta}(\emptyset)$.

5.1 LCFRS/sDCP Hybrid Grammars

We first couple a LCFRS and a sDCP in order to describe a set of hybrid trees.

A **LCFRS/sDCP hybrid grammar** (over Γ and Σ) is a tuple $G = (N, S, (\Gamma, \Sigma), P)$, subject to the following restrictions. The objects Γ and Σ are ranked alphabets with $\Gamma = \Gamma^{(0)}$ and $\Sigma = \Sigma^{(1)}$. As mere sets of symbols, we demand $\Gamma \subseteq \Sigma$. Let Δ be the ranked alphabet $\Sigma \setminus \Gamma$, with $\mathrm{rk}_\Delta(\delta) = \mathrm{rk}_\Sigma(\delta) = 1$ for $\delta \in \Delta$. The set P is a finite set of **hybrid rules**, each of the form:

$$[A(s^{(1)}_{1,k}) \to r_1,\ A(s^{(2)}_{1,m}) \to r_2] \tag{4}$$

where $\mathcal{D}(A(s^{(1)}_{1,k})) \to \mathcal{D}(r_1)$ satisfies the syntactic constraints of an individual LCFRS rule and $\mathcal{D}(A(s^{(2)}_{1,m})) \to \mathcal{D}(r_2)$ satisfies the syntactic constraints of a sDCP rule. The added indices occur at nonterminals in r_1 and r_2 and at terminals from Γ in $s^{(1)}_{1,k}$ and in $s^{(2)}_{1,m}$ and r_2. We require that each index in a hybrid rule either couples a pair of identical terminals or couples a pair of identical nonterminals. Let P_1 be the set of all $\mathcal{D}(A(s^{(1)}_{1,k})) \to \mathcal{D}(r_1)$, where $A(s^{(1)}_{1,k}) \to r_1$ occurs as the first component of a hybrid rule as in Equation (4). The set P_2 is similarly defined, taking the second components. We now further require that (N, S, Γ, P_1) is a LCFRS and (N, S, Σ, P_2) is a sDCP. We refer to these two grammars as the first and second **components**, respectively, of G.

In order to define the "derives" relation $\Rightarrow^{u,\rho}_G$, for some index u and some rule ρ of the form of Equation (4), we need the additional notions of nonterminal reindexing and of terminal reindexing. The **nonterminal reindexing** is an injective function f_U that replaces each index at a nonterminal occurrence of the rule by one that does not clash with the indices in an existing set $U \subseteq \mathbb{N}_+$. We may, for example, define f_U such that it maps each $v \in \mathbb{N}_+$ to the smallest $v' \in \mathbb{N}_+$ such that $v' \notin U \cup \{f_U(1), \ldots, f_U(v-1)\}$. We

extend f_U to apply to terms, s-terms, and rules in a natural way, to replace indices by other indices, but leaving all other symbols unaffected. A **terminal reindexing** g maps indices at occurrences of terminals in the rule to the indices of corresponding occurrences of terminals in the sentential form; we extend g to terms, s-terms, and rules in the same way as for f_U. The definition of f_U is fixed while an appropriate g needs to be chosen for each derivation step.

We can now formally define:

$$[s_1 \cdot \langle A^{\boxed{u}}(s'^{(1)}_{1,k})\rangle \cdot s'_1,\ s_2 \cdot \langle A^{\boxed{u}}(s'^{(2)}_{1,k})\rangle \cdot s'_2] \ \Rightarrow^{u,\rho}_{G} \ [s_1 \cdot r'_1 \cdot s'_1,\ s_2 \cdot r'_2 \cdot s'_2]$$

for every $s_1, s'_1 \in \mathcal{I}^*_{N\cup\Gamma,\emptyset}$, sequence $s'^{(1)}_{1,k}$ of s-terms in $\mathcal{I}^*_{\Gamma,\emptyset}$, and $s_2, s'_2 \in \mathcal{I}^*_{N\cup\Gamma,\Delta}$ and sequence $s'^{(2)}_{1,k}$ of s-terms in $\mathcal{I}^*_{\Gamma,\Delta}$, if:

- $\rho \in P$ is $[A(s^{(1)}_{1,k}) \to r_1, A(s^{(2)}_{1,k}) \to r_2]$,
- $U = ind(s_1 \cdot \langle A^{\boxed{u}}(s'^{(1)}_{1,k})\rangle \cdot s'_1) \setminus \{u\} = ind(s_2 \cdot \langle A^{\boxed{u}}(s'^{(2)}_{1,k})\rangle \cdot s'_2) \setminus \{u\}$,
- there is a terminal reindexing g such that $g(s^{(1)}_{1,k}) = s'^{(1)}_{1,k}$ (and hence $g(s^{(2)}_{1,k}) = s'^{(2)}_{1,k}$),
- $A(s'^{(1)}_{1,k}) \to r'_1$ is obtained from $g(f_U(A(s^{(1)}_{1,k}) \to r_1))$ by consistently substituting occurrences of variables by s-terms in $\mathcal{I}^*_{\Gamma,\emptyset}$, and $A(s'^{(2)}_{1,k}) \to r'_2$ is obtained from $g(f_U(A(s^{(2)}_{1,k}) \to r_2))$ by consistently substituting occurrences of variables by s-terms in $\mathcal{I}^*_{\Gamma,\Delta}$.

Example 10
Let us consider the rule:

$$\rho:\ [A(x_1\mathbf{a}^{\boxed{3}}x_2) \to B^{\boxed{1}}(x_1)\,C^{\boxed{2}}(x_2),\ A(\mathbf{d}(\mathbf{a}^{\boxed{3}}x_1x_2)) \to B^{\boxed{1}}(x_1)\,C^{\boxed{2}}(x_2)]$$

of some LCFRS/sDCP hybrid grammar and the sentential form:

$$[D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ \underline{A^{\boxed{1}}(\mathbf{b}^{\boxed{4}}\mathbf{a}^{\boxed{5}}\mathbf{c}^{\boxed{6}})},\ \ D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ \underline{A^{\boxed{1}}(\mathbf{d}(\mathbf{a}^{\boxed{5}}\mathbf{b}^{\boxed{4}}\mathbf{c}^{\boxed{6}}))}]$$

We apply ρ to the underlined subterms, that is, $u = 1$. We have:

$$U = ind(D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ A^{\boxed{1}}(\mathbf{b}^{\boxed{4}}\mathbf{a}^{\boxed{5}}\mathbf{c}^{\boxed{6}})) \setminus \{1\} = \{2,3,4,5,6\}$$

We define the nonterminal reindexing such that $f_U(1) = 1$ and $f_U(2) = 7$. As terminal reindexing we use g such that $g(3) = 5$. Thus we obtain the reindexed rule:

$$g(f_U(\rho)) = [A(x_1\mathbf{a}^{\boxed{5}}x_2) \to B^{\boxed{1}}(x_1)\,C^{\boxed{7}}(x_2),\ A(\mathbf{d}(\mathbf{a}^{\boxed{5}}x_1x_2)) \to B^{\boxed{1}}(x_1)\,C^{\boxed{7}}(x_2)]$$

By using the substitutions $x_1 \mapsto \mathbf{b}^{\boxed{4}}$ and $x_2 \mapsto \mathbf{c}^{\boxed{6}}$ for both components we obtain the derivation step:

$$\begin{aligned} & [D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ \underline{A^{\boxed{1}}(\mathbf{b}^{\boxed{4}}\mathbf{a}^{\boxed{5}}\mathbf{c}^{\boxed{6}})},\ D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ \underline{A^{\boxed{1}}(\mathbf{d}(\mathbf{a}^{\boxed{5}}\mathbf{b}^{\boxed{4}}\mathbf{c}^{\boxed{6}}))}] \\ \Rightarrow_G^{1,\rho} \quad & [D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ B^{\boxed{1}}(\mathbf{b}^{\boxed{4}})\ C^{\boxed{7}}(\mathbf{c}^{\boxed{6}}),\ D^{\boxed{2}}(\mathbf{a}^{\boxed{3}})\ B^{\boxed{1}}(\mathbf{b}^{\boxed{4}})\ C^{\boxed{7}}(\mathbf{c}^{\boxed{6}})] \end{aligned}$$ ■

We write $\Rightarrow_G$ for $\bigcup_{u\in\mathbb{N}_+,\rho\in P} \Rightarrow_G^{u,\rho}$. The hybrid language induced by G is:

$$[G] = \{[s_1, s_2] \in \mathcal{I}^*_{\Gamma,\emptyset} \times \mathcal{I}^*_{\Gamma,\Delta} \mid [\langle S^{\boxed{1}}(s_1)\rangle, \langle S^{\boxed{1}}(s_2)\rangle] \Rightarrow_G^* [\langle\rangle, \langle\rangle]\} \tag{5}$$

Note that apart from the indices, the first component of a pair $[s_1, s_2] \in [G]$ consists of a string of terminals from Γ and the second is a s-term built up of terminals in $\Gamma \cup \Delta$. Moreover, every occurrence of $\gamma \in \Gamma$ in s_1 corresponds to exactly one in s_2 and vice versa, because of the common indices.

From a pair $[s_1, s_2] \in [G]$, we can construct the hybrid tree $(s, \leq_s)$ over (Γ, Σ) by letting $s = \mathcal{D}(s_2)$ and, for each combination of positions p_1, p_1', p_2, p_2' such that $s_1(p_1) = s_2(p_2) \in \mathcal{I}(\Gamma)$ and $s_1(p_1') = s_2(p_2') \in \mathcal{I}(\Gamma)$, we set $p_2 \leq_s p_2'$ if and only if $p_1 \leq_\ell p_1'$. (The lexicographical ordering $\leq_\ell$ on positions here simplifies to the linear ordering of integers, as positions in strings always have length 1.) In words, occurrences of terminals in s obtain a total order in accordance with the order in which corresponding terminals occur in s_1. The set of all such $(s, \leq_s)$ will be denoted by $L(G)$.

Example 11
Abbreviating each German word by its first letter, the hybrid tree in Figure 4a is obtained by the following LCFRS/sDCP hybrid grammar G. (All arguments in the second component are synthesized.)

$$\rho_1 : [VP(x_1x_2x_3) \to V^{\boxed{1}}(x_1, x_3)\ ADV^{\boxed{2}}(x_2), VP(\mathbf{VP}(x_1x_2)) \to V^{\boxed{1}}(x_1)\ ADV^{\boxed{2}}(x_2)]$$
$$\rho_2 : [V(\mathbf{h}^{\boxed{1}}, \mathbf{g}^{\boxed{2}}) \to \varepsilon, V(\mathbf{V}(\mathbf{h}^{\boxed{1}}\mathbf{g}^{\boxed{2}})) \to \varepsilon]$$
$$\rho_3 : [ADV(\mathbf{s}^{\boxed{1}}) \to \varepsilon, ADV(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \to \varepsilon]$$

We derive:

$$\begin{aligned} & [VP^{\boxed{1}}(\mathbf{h}^{\boxed{2}}\mathbf{s}^{\boxed{3}}\mathbf{g}^{\boxed{4}}),\ VP^{\boxed{1}}(\mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}\mathbf{g}^{\boxed{4}})\ \mathbf{ADV}(\mathbf{s}^{\boxed{3}})))] \\ \Rightarrow_G^{1,\rho_1}\ & [V^{\boxed{1}}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}})\ ADV^{\boxed{5}}(\mathbf{s}^{\boxed{3}}),\ V^{\boxed{1}}(\mathbf{V}(\mathbf{h}^{\boxed{2}}\mathbf{g}^{\boxed{4}}))\ ADV^{\boxed{5}}(\mathbf{ADV}(\mathbf{s}^{\boxed{3}}))] \\ \Rightarrow_G^{1,\rho_2}\ & [ADV^{\boxed{5}}(\mathbf{s}^{\boxed{3}}),\ ADV^{\boxed{5}}(\mathbf{ADV}(\mathbf{s}^{\boxed{3}}))] \\ \Rightarrow_G^{5,\rho_3}\ & [\varepsilon, \varepsilon] \end{aligned}$$

where we have used the following reindexing functions:

application of ...	nonterminal reindexing	terminal reindexing
ρ_1	$f_{\{2,3,4\}}(2) = 5$	g identity
ρ_2	$f_{\{2,3,4,5\}}$ identity	$g(1) = 2$ and $g(2) = 4$
ρ_3	$f_{\{3\}}$ identity	$g(1) = 3$

and $f_U(i) = i$ and $g(j) = j$ if not specified otherwise.

Note that in the LCFRS that is the first component, nonterminal V has fanout 2, and the LCFRS thereby has fanout 2. The tree produced by the second component is a parse tree in the traditional sense—that is, it specifies exactly how the first component analyzes the input string. Each LCFRS can in fact be extended to become a canonical LCFRS/sDCP hybrid of this kind, in which there is no freedom in the coupling of the string grammar and the tree grammar. Traditional frameworks for LCFRS parsing can be reinterpreted as using such hybrid grammars.

To give a first illustration of the additional freedom offered by LCFRS/sDCP hybrid grammars, consider the following alternative, where the first and second components have a different structure. It derives the same hybrid tree, but now with all nonterminals in the first component having fanout 1, by which we obtain a syntactic variant of a context-free grammar. This, however, requires the second component to have a less straightforward form:

$$[VP(x_1) \rightarrow V^{\boxed{1}}(x_1),\ VP(\mathbf{VP}(x_1)) \rightarrow V^{\boxed{1}}(x_1)]$$
$$[V(\mathbf{h}^{\boxed{1}} x_1 \mathbf{g}^{\boxed{2}}) \rightarrow ADV^{\boxed{3}}(x_1),\ V(\mathbf{V}(\mathbf{h}^{\boxed{1}} \mathbf{g}^{\boxed{2}})\ x_1) \rightarrow ADV^{\boxed{3}}(x_1)]$$
$$[ADV(\mathbf{s}^{\boxed{1}}) \rightarrow \varepsilon,\ ADV(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \rightarrow \varepsilon] \quad \blacksquare$$

A derivation of a LCFRS/sDCP hybrid grammar G can be represented by a **derivation tree** (cf. Figure 8). It combines a derivation tree of the first component of G and a derivation tree of its second component, letting them share the common parts. We make a graphical distinction between arguments of the first component (rectangles) and those of the second component (ovals). Implicit in a derivation tree is the reindexing of terminal indices. The evaluation ϕ_1 of the derivation tree d in Figure 8 yields the indexed string $\phi_1(d) = \mathbf{h}^{\boxed{2}}\mathbf{s}^{\boxed{3}}\mathbf{g}^{\boxed{4}}$ and the evaluation ϕ_2 of d yields the tree $\phi_2(d) = \mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}\mathbf{g}^{\boxed{4}})\ \mathbf{ADV}(\mathbf{s}^{\boxed{3}}))$.

Example 12
The hybrid tree in Figure 4b can be obtained by the following LCFRS/sDCP hybrid grammar. In the second component, T (for transitive verb) has two inherited arguments, for the subject and the object, whereas I (intransitive verb) has one inherited argument,

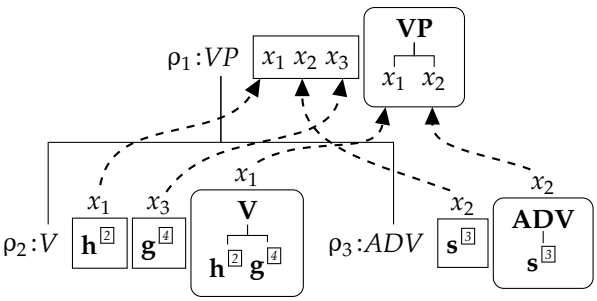

Figure 8
The derivation tree corresponding to the derivation shown in Example 11.

for the subject. (To keep the example simple, we conflate different verb forms, so that the grammar overgenerates.)

$$[S(x_1x_2) \to S_2^{\boxed{1}}(x_1, x_2)\ ,\ S(x_1) \to S_2^{\boxed{1}}(x_1)]$$
$$[S_2(x_1x_3, x_2x_4) \to N^{\boxed{1}}(x_1)T^{\boxed{2}}(x_2)S_2^{\boxed{3}}(x_3, x_4)\ ,\ S_2(x_3) \to N^{\boxed{1}}(x_1)T^{\boxed{2}}(x_1, x_2, x_3)S_2^{\boxed{3}}(x_2)]$$
$$[S_2(x_1, x_2) \to N^{\boxed{1}}(x_1)I^{\boxed{2}}(x_2)\ ,\ S_2(x_2) \to N^{\boxed{1}}(x_1)I^{\boxed{2}}(x_1, x_2)]$$
$$[T(\mathbf{zag}^{\boxed{1}}) \to \varepsilon\ ,\ T(x_1, x_2, \mathbf{zag}^{\boxed{1}}(x_1x_2)) \to \varepsilon]$$
$$[T(\mathbf{helpen}^{\boxed{1}}) \to \varepsilon\ ,\ T(x_1, x_2, \mathbf{helpen}^{\boxed{1}}(x_1x_2)) \to \varepsilon]$$
$$[I(\mathbf{lezen}^{\boxed{1}}) \to \varepsilon\ ,\ I(x_1, \mathbf{lezen}^{\boxed{1}}(x_1)) \to \varepsilon]$$
$$[N(\mathbf{Jan}^{\boxed{1}}) \to \varepsilon\ ,\ N(\mathbf{Jan}^{\boxed{1}}) \to \varepsilon],\ \textit{etc.}$$

We derive (abbreviating each Dutch word by its first letter):

$$\begin{aligned}
&[S^{\boxed{1}}(\mathbf{J}^{\boxed{2}}\mathbf{P}^{\boxed{3}}\mathbf{M}^{\boxed{4}}\mathbf{z}^{\boxed{5}}\mathbf{h}^{\boxed{6}}\mathbf{l}^{\boxed{7}})\ ,\ S^{\boxed{1}}(\mathbf{z}^{\boxed{5}}(\mathbf{J}^{\boxed{2}}\mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}))))]\\
&\Rightarrow\ [S_2^{\boxed{1}}(\mathbf{J}^{\boxed{2}}\mathbf{P}^{\boxed{3}}\mathbf{M}^{\boxed{4}}, \mathbf{z}^{\boxed{5}}\mathbf{h}^{\boxed{6}}\mathbf{l}^{\boxed{7}})\ ,\ S_2^{\boxed{1}}(\mathbf{z}^{\boxed{5}}(\mathbf{J}^{\boxed{2}}\mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}))))]\\
&\Rightarrow\ [N^{\boxed{8}}(\mathbf{J}^{\boxed{2}})T^{\boxed{9}}(\mathbf{z}^{\boxed{5}})S_2^{\boxed{1}}(\mathbf{P}^{\boxed{3}}\mathbf{M}^{\boxed{4}}, \mathbf{h}^{\boxed{6}}\mathbf{l}^{\boxed{7}})\ ,\\
&\quad N^{\boxed{8}}(\mathbf{J}^{\boxed{2}})T^{\boxed{9}}(\mathbf{J}^{\boxed{2}}, \mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}})), \mathbf{z}^{\boxed{5}}(\mathbf{J}^{\boxed{2}}\mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}))))S_2^{\boxed{1}}(\mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}})))]\\
&\Rightarrow^2\ [S_2^{\boxed{1}}(\mathbf{P}^{\boxed{3}}\mathbf{M}^{\boxed{4}}, \mathbf{h}^{\boxed{6}}\mathbf{l}^{\boxed{7}})\ ,\ S_2^{\boxed{1}}(\mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}})))]\\
&\Rightarrow\ [N^{\boxed{8}}(\mathbf{P}^{\boxed{3}})T^{\boxed{9}}(\mathbf{h}^{\boxed{6}})S_2^{\boxed{1}}(\mathbf{M}^{\boxed{4}}, \mathbf{l}^{\boxed{7}})\ ,\\
&\quad N^{\boxed{8}}(\mathbf{P}^{\boxed{3}})T^{\boxed{9}}(\mathbf{P}^{\boxed{3}}, \mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}), \mathbf{h}^{\boxed{6}}(\mathbf{P}^{\boxed{3}}\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}})))S_2^{\boxed{1}}(\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}))]\\
&\Rightarrow^2\ [S_2^{\boxed{1}}(\mathbf{M}^{\boxed{4}}, \mathbf{l}^{\boxed{7}})\ ,\ S_2^{\boxed{1}}(\mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}))]\\
&\Rightarrow\ [N^{\boxed{8}}(\mathbf{M}^{\boxed{4}})I^{\boxed{9}}(\mathbf{l}^{\boxed{7}})\ ,\ N^{\boxed{8}}(\mathbf{M}^{\boxed{4}})I^{\boxed{9}}(\mathbf{M}^{\boxed{4}}, \mathbf{l}^{\boxed{7}}(\mathbf{M}^{\boxed{4}}))]\ \Rightarrow^2\ \varepsilon
\end{aligned}$$

■

5.2 Other Classes of Hybrid Grammars

In order to illustrate the generality of our framework, we will sketch three more classes of hybrid grammars. In these three classes, the first component or the second component, or both, are less powerful than in the case of the LCFRS/sDCP hybrid grammars defined previously, and thereby the resulting hybrid grammars are less powerful, in the light of the observation that sMGs are syntactic variants of well-nested LCFRSs and sCFTGs are syntactic variants of sDCPs with s-rank restricted to 1. Noteworthy are the differences between the four classes of hybrid grammars in the formal definition of their derivations.

A **sMG/sCFTG hybrid grammar** (over Γ and Σ) is a tuple $G = (N, S, (\Gamma, \Sigma), P)$, where Γ and Σ are as in the case of LCFRS/sDCP hybrid grammars. The hybrid rules in P are now of the form:

$$[A(x_{1,k}) \to r_1, A(x_{1,k}) \to r_2] \tag{6}$$

where $A(x_{1,k}) \to \mathcal{D}(r_1)$ is a sMG rule and $A(x_{1,k}) \to \mathcal{D}(r_2)$ is a sCFTG rule.

In the definition of the "derives" relation $\Rightarrow_G^{u,\rho}$ we have to use a reindexing function. Because terminals are produced by a rule application (instead of being consumed as in the LCFRS/sDCP case), there is no need for a terminal reindexing that matches indices

of terminal occurrences of the applied rule with those of the sentential form. Instead, terminal indices occurring in the rule have to be reindexed away from the sentential form in the same way as for nonterminal indices. Thus we use one reindexing function f_U that applies to nonterminal indices and terminal indices.

We define $[s_1, s_2] \Rightarrow_G^{u,\rho} [s_1', s_2']$ for every $s_1, s_1' \in \mathcal{I}^*_{N \cup \Gamma, \emptyset}$ and $s_2, s_2' \in \mathcal{I}^*_{N \cup \Gamma, \Delta}$ if and only if:

- there are positions p_1 and p_2 such that $s_1(p_1) = A^{\boxed{u}}$ and $s_2(p_2) = A^{\boxed{u}}$,
- $\rho \in P$ is $[A(x_{1,k}) \to r_1, A(x_{1,k}) \to r_2]$,
- $U = ind(s_1) \setminus \{u\} = ind(s_2) \setminus \{u\}$,
- $s_i' = s_i[\![f_U(r_i)]\!]_{p_i}$ for $i = 1, 2$.

We write $\Rightarrow_G$ for $\bigcup_{u \in \mathbb{N}_+, \rho \in P} \Rightarrow_G^{u,\rho}$ and define the hybrid language induced by G as:

$$[G] = \{[s_1, s_2] \in \mathcal{I}^*_{\Gamma,\emptyset} \times \mathcal{I}^*_{\Gamma,\Delta} \mid [\langle S^{\boxed{1}} \rangle, \langle S^{\boxed{1}} \rangle] \Rightarrow_G^* [s_1, s_2]\} \tag{7}$$

As before, this defines a set $L(G)$ of hybrid trees. Note the structural difference between Equations (5) and (7).

Example 13
The hybrid tree in Figure 4a is obtained by the following sMG/sCFTG hybrid grammar:

$$[VP \to V^{\boxed{1}}(ADV^{\boxed{2}}) \ , \ VP \to \mathbf{VP}(V^{\boxed{1}} ADV^{\boxed{2}})]$$
$$[V(x_1) \to \mathbf{h}^{\boxed{1}} x_1 \mathbf{g}^{\boxed{2}} \ , \ V \to \mathbf{V}(\mathbf{h}^{\boxed{1}} \mathbf{g}^{\boxed{2}})]$$
$$[ADV \to \mathbf{s}^{\boxed{1}} \ , \ ADV \to \mathbf{ADV}(\mathbf{s}^{\boxed{1}})]$$

We derive:

$$\begin{aligned}[VP^{\boxed{1}}, VP^{\boxed{1}}] &\Rightarrow [V^{\boxed{1}}(ADV^{\boxed{2}}), \mathbf{VP}(V^{\boxed{1}} ADV^{\boxed{2}})] \\ &\Rightarrow [\mathbf{h}^{\boxed{1}} ADV^{\boxed{2}} \mathbf{g}^{\boxed{3}}, \mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{1}} \mathbf{g}^{\boxed{3}}) \ ADV^{\boxed{2}})] \\ &\Rightarrow [\mathbf{h}^{\boxed{1}} \mathbf{s}^{\boxed{2}} \mathbf{g}^{\boxed{3}}, \mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{1}} \mathbf{g}^{\boxed{3}}) \ \mathbf{ADV}(\mathbf{s}^{\boxed{2}}))]\end{aligned}$$

As in Example 11, there is an alternative hybrid grammar in which all nonterminals in the first component have rank 0, which is thereby context-free. This is at the expense of a more complicated second component. ■

Example 14
Hybrid trees for cross-serial dependencies as in Figure 5 can be obtained through the following sMG/sCFTG hybrid grammar, with start symbol A:

$$[A \to S^{\boxed{1}}(\varepsilon) \ , \ A \to S^{\boxed{1}}]$$
$$[S(x_1) \to \mathbf{a}^{\boxed{1}} \ S^{\boxed{2}}(x_1 \ \mathbf{b}^{\boxed{3}}) \ , \ S \to \mathbf{S}(\mathbf{a}^{\boxed{1}} S^{\boxed{2}} \mathbf{b}^{\boxed{3}})]$$
$$[S(x_1) \to x_1 \ , \ S \to \varepsilon]$$

One can derive, for instance:

$$[A^{\boxed{1}}, A^{\boxed{1}}] \Rightarrow^* [\mathbf{a}^{\boxed{1}} \mathbf{a}^{\boxed{2}} \mathbf{b}^{\boxed{3}} \mathbf{b}^{\boxed{5}}, \mathbf{S}(\mathbf{a}^{\boxed{1}} \ \mathbf{S}(\mathbf{a}^{\boxed{2}} \mathbf{b}^{\boxed{5}}) \ \mathbf{b}^{\boxed{3}})]$$ ■

Analogously to LCFRS/sDCP and sMG/sCFTG hybrid grammars, one can define **LCFRS/sCFTG hybrid grammars** and **sMG/sDCP hybrid grammars**, with suitable definitions of $\Rightarrow_G$ obtained straightforwardly by combining elements from the earlier definitions. The hybrid language induced by LCFRS/sCFTG hybrid grammar G is:

$$[G] = \{[s_1, s_2] \in \mathcal{I}^*_{\Gamma,\emptyset} \times \mathcal{I}^*_{\Gamma,\Delta} \mid [\langle S^{\boxed{1}}(s_1)\rangle, \langle S^{\boxed{1}}\rangle] \Rightarrow^*_G [\langle\rangle, s_2]\}$$

and the hybrid language induced by sMG/sDCP hybrid grammar G is:

$$[G] = \{[s_1, s_2] \in \mathcal{I}^*_{\Gamma,\emptyset} \times \mathcal{I}^*_{\Gamma,\Delta} \mid [\langle S^{\boxed{1}}\rangle, \langle S^{\boxed{1}}(s_2)\rangle] \Rightarrow^*_G [s_1, \langle\rangle]\}$$

Example 15
The hybrid tree in Figure 4a is obtained by the LCFRS/sCFTG hybrid grammar:

$$[VP(x_1x_2x_3) \to V^{\boxed{1}}(x_1, x_3)\ ADV^{\boxed{2}}(x_2)\ \ ,\ \ VP \to \mathbf{VP}(V^{\boxed{1}} ADV^{\boxed{2}})]$$
$$[V(\mathbf{h}^{\boxed{1}}, \mathbf{g}^{\boxed{2}}) \to \varepsilon\ \ ,\ \ V \to \mathbf{V}(\mathbf{h}^{\boxed{1}}\mathbf{g}^{\boxed{2}})]$$
$$[ADV(\mathbf{s}^{\boxed{1}}) \to \varepsilon\ \ ,\ \ ADV \to \mathbf{ADV}(\mathbf{s}^{\boxed{1}})]$$

The derivation of the hybrid tree in Figure 4a is:

$$\begin{aligned}
&[VP^{\boxed{1}}(\mathbf{h}^{\boxed{2}}\mathbf{s}^{\boxed{3}}\mathbf{g}^{\boxed{4}}),\ VP^{\boxed{1}}]\\
&\Rightarrow [V^{\boxed{1}}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}})\ ADV^{\boxed{5}}(\mathbf{s}^{\boxed{3}}),\ \mathbf{VP}(V^{\boxed{1}}\ ADV^{\boxed{5}})]\\
&\Rightarrow [ADV^{\boxed{5}}(\mathbf{s}^{\boxed{3}}),\ \mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}})\ ADV^{\boxed{5}})]\\
&\Rightarrow [\varepsilon,\ \mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}})\ \mathbf{ADV}(\mathbf{s}^{\boxed{3}}))]
\end{aligned}$$

A sMG/sDCP hybrid grammar that achieves the same is:

$$[VP \to V^{\boxed{1}}(ADV^{\boxed{2}})\ \ ,\ \ VP(\mathbf{VP}(x_1x_2)) \to V^{\boxed{1}}(x_1)\ ADV^{\boxed{2}}(x_2)]$$
$$[V(x_1) \to \mathbf{h}^{\boxed{1}}x_1\mathbf{g}^{\boxed{2}}\ \ ,\ \ V(\mathbf{V}(\mathbf{h}^{\boxed{1}}\mathbf{g}^{\boxed{2}})) \to \varepsilon]$$
$$[ADV \to \mathbf{s}^{\boxed{1}}\ \ ,\ \ ADV(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \to \varepsilon]$$

We can derive:

$$\begin{aligned}
&[VP^{\boxed{1}},\ VP^{\boxed{1}}(\mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}})\ \mathbf{ADV}(\mathbf{s}^{\boxed{3}})))]\\
&\Rightarrow [V^{\boxed{1}}(ADV^{\boxed{5}}),\ V^{\boxed{1}}(\mathbf{V}(\mathbf{h}^{\boxed{2}}, \mathbf{g}^{\boxed{4}}))\ ADV^{\boxed{5}}(\mathbf{ADV}(\mathbf{s}^{\boxed{3}}))]\\
&\Rightarrow [\mathbf{h}^{\boxed{2}}\ ADV^{\boxed{5}}\ \mathbf{g}^{\boxed{4}},\ ADV^{\boxed{5}}(\mathbf{ADV}(\mathbf{s}^{\boxed{3}}))]\\
&\Rightarrow [\mathbf{h}^{\boxed{2}}\mathbf{s}^{\boxed{3}}\mathbf{g}^{\boxed{4}},\ \varepsilon]
\end{aligned}$$

■

5.3 Probabilistic LCFRS/sDCP Hybrid Grammars and Parsing

In the usual way, we can extend LCFRS/sDCP hybrid grammars to become probabilistic LCFRS/sDCP hybrid grammars. For this we can assign a probability to each hybrid rule,

Algorithm 1 Parsing a string with probabilistic LCFRS/sDCP hybrid grammars

Input: probabilistic LCFRS/sDCP hybrid grammar G over Γ and Σ
$w = \alpha_1 \cdots \alpha_n \in \Gamma^*$
Output: hybrid tree h with $\mathrm{str}(h) = w$ and with most probable derivation

1: extract the first component of G; resulting in a probabilistic LCFRS G'
2: parse w according to G' with any standard LCFRS parser; resulting in a most likely derivation tree d
3: enrich the dependency graph of d by the arguments which correspond to the second components of the applied LCFRS/sDCP rules; resulting in the intermediate structure $\hat{d}$
4: choose a terminal indexing of w; resulting in $s_1 \in \mathcal{I}^*_{\Gamma,\emptyset}$
5: traverse $\hat{d}$ and reindex terminal indices to match those in s_1; resulting in derivation tree d'
6: **return** hybrid tree h corresponding to $[s_1, \phi_2(d')]$

under the constraint of a **properness** condition. More precisely, for each nonterminal A the probabilities of all hybrid rules with A in their left-hand sides sum up to 1. In this way a probabilistic LCFRS/sDCP hybrid grammar induces a distribution over hybrid trees. Thus it can be considered as a generative model. See also Nivre (2010) for a general survey of probabilistic parsing.

Algorithm 1 shows a parsing pipeline which takes as input a probabilistic LCFRS/sDCP hybrid grammar G and a sentence $w \in \Gamma^*$. As output it computes the hybrid tree $h \in L(G)$ that is derived by the most likely derivation whose first component derives w. In line 1 the first component of G is extracted. Because we later will restore the second component, we assume that a tag is attached to each LCFRS rule that uniquely identifies the original LCFRS/sDCP rule. This also means that two otherwise identical LCFRS rules are treated as distinct if they were taken from two different LCFRS/sDCP rules. In line 2 the string w is parsed. For this any standard LCFRS parser can be used. Such parsers, for instance those by Seki et al. (1991) and Kallmeyer (2010), typically run in polynomial time. The technical details of the chosen LCFRS parser (like the used form of items or the form of iteration over rules) are irrelevant, as for our framework only the functionality of the parsing component matters. The parsing algorithm builds a succinct representation of all derivations of w. In a second, linear-time phase the most likely derivation tree d for w is extracted. In line 3 the dependency graph of d is enriched by the inherited and synthesized arguments that correspond to the second components of the rules occurring in d; here, the identity of the original LCFRS/sDCP rule is needed. This results in an intermediate structure $\hat{d}$. This is not yet a derivation tree of G, as the terminals still need to be assigned unique indices. This is done first by an indexing of the terminals from w in an arbitrary manner, and then by traversing the derivation to associate these indices with the appropriate terminal occurrences in the derivation, leading to a derivation tree d' of the LCFRS/sDCP hybrid grammar G. Note that $\phi_1(d') = s_1$. Finally, in line 6 the derivation tree d' is evaluated by ϕ_2 yielding a tree in $\mathcal{I}^*_{\Gamma,\Delta}$.

We illustrate Algorithm 1 by means of the (non-probabilistic) LCFRS/sDCP hybrid grammar G from Example 11 and the sentence $w = \mathbf{h\ s\ g}$. Figure 9 shows d, $\hat{d}$, d', and

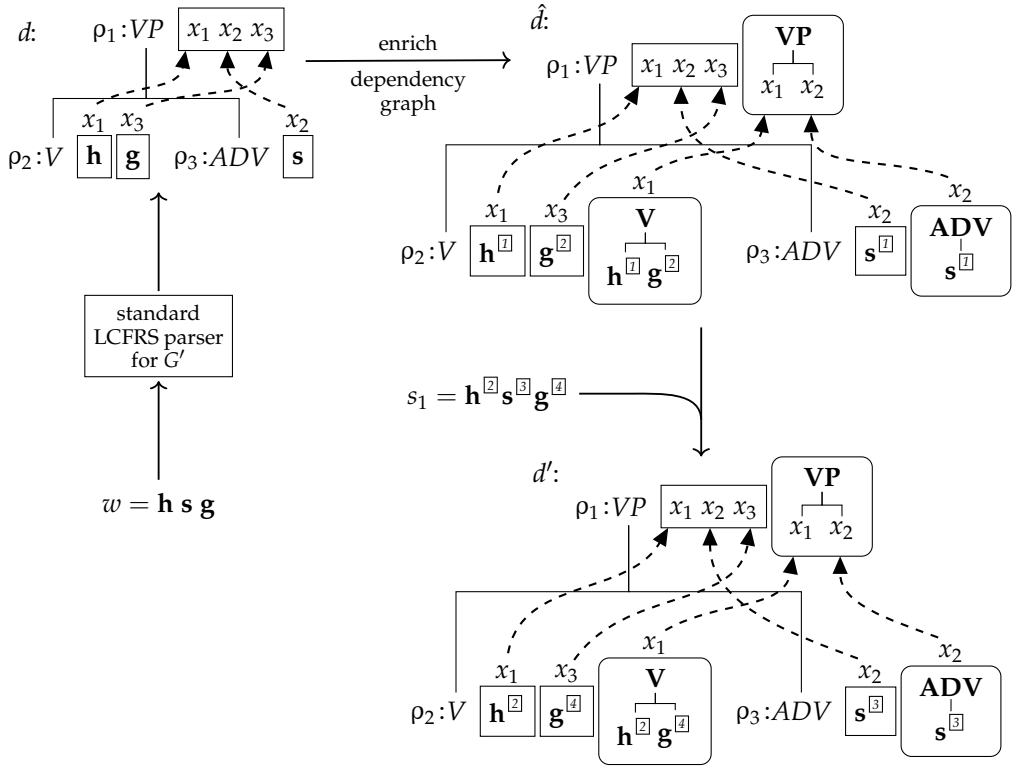

Figure 9
Algorithm 1 applied to the LCFRS/sDCP hybrid grammar G from Example 11 and the sentence $w = \mathbf{h}\ \mathbf{s}\ \mathbf{g}$.

s_1; as part of d' we obtain $\phi_2(d') = \mathbf{VP}(\mathbf{V}(\mathbf{h}^{\boxed{2}}\mathbf{g}^{\boxed{4}})\ \mathbf{ADV}(\mathbf{s}^{\boxed{3}}))$. The LCFRS grammar G' resulting from the extraction in line 1 is:

$$\rho_1 :\ VP(x_1x_2x_3) \to V(x_1, x_3)\ ADV(x_2) \quad \rho_2 :\ V(\mathbf{h}, \mathbf{g}) \to \varepsilon \quad \rho_3 :\ ADV(\mathbf{s}) \to \varepsilon$$

6. Grammar Induction

For most practical applications, hybrid grammars would not be written by hand, but would be automatically extracted from finite corpora of hybrid trees, over which they should generalize. To be precise, the task is to construct a hybrid grammar G out of a corpus c such that $c \subseteq L(G)$.

During grammar induction each hybrid tree $h = (s, \leq_s)$ of the corpus is decomposed. A decomposition will determine the structure of a derivation of h by the grammar. Classically, this decomposition and thereby the resulting derivations resemble the structure of s. This approach has been pursued, for instance, by Charniak (1996) for CFGs, and by Maier and Søgaard (2008) and Kuhlmann and Satta (2009) for LCFRSs. There is no guarantee, however, that this approach is optimal from the perspective of, for example, parsing efficiency, the grammar's potential to generalize from training data, or the size of the grammar.

For a more general approach that offers additional degrees of freedom we extend the framework of Nederhof and Vogler (2014) and let grammar induction depend on a decomposition strategy of the string $\mathrm{str}(h)$, called recursive partitioning. One may choose one out of several such strategies. We consider four instances of induction of hybrid grammars, which vary in the type of hybrid trees that occur in the corpus, and the type of the resulting hybrid grammar:

	corpus	hybrid grammar
1.	phrase structures	LCFRS/sDCP
2.	phrase structures	LCFRS/sCFTG
3.	dependency structures	LCFRS/sDCP
4.	dependency structures	LCFRS/sCFTG

6.1 Recursive Partitioning and Induction of LCFRS

A **recursive partitioning** of a string w of length n is a tree π whose nodes are labeled with subsets of $[n]$. The root of π is labeled with $[n]$. Each leaf of π is labeled with a singleton subset of $[n]$. Each non-leaf node has at least two children and is labeled with the union of the labels of its children, which furthermore must be disjoint. To fit recursive partitionings into our framework of s-terms, we regard $\mathcal{P}([n])$ as ranked alphabet with $\mathcal{P}([n]) = \mathcal{P}([n])^{(1)}$. We let $\pi \in T^*_{\mathcal{P}([n])}$ with $|\pi| = 1$.

We say a set $J \subseteq [n]$ has **fanout** k if k is the smallest number such that J can be written as $J = J_1 \cup \ldots \cup J_k$, where:

- each J_ℓ ($\ell \in [k]$) is of the form $\{i_\ell, i_\ell + 1, \ldots, i_\ell + m_\ell\}$, for some i_ℓ and $m_\ell \geq 0$, and
- $i \in J_\ell$, $i' \in J_{\ell'}$ ($\ell, \ell' \in [k]$) and $\ell < \ell'$ imply $i < i'$.

Note that $J_1, \ldots, J_k$ are uniquely defined by this, and we write $\mathrm{spans}(J) = \langle J_1, \ldots, J_k \rangle$. The fanout of a recursive partitioning is defined as the maximal fanout of its nodes.

Figure 10 presents two recursive partitionings of a string with seven positions. The left one has fanout 3 (because of the node label $\{1,3,6,7\} = \{1\} \cup \{3\} \cup \{6,7\}$), whereas the right one has fanout 2.

Algorithm 2 constructs a LCFRS G out of a string $w = \alpha_1 \cdots \alpha_n$ and a recursive partitioning π of w. The nonterminals are symbols of the form $(\!|J|\!)$, where J is a node

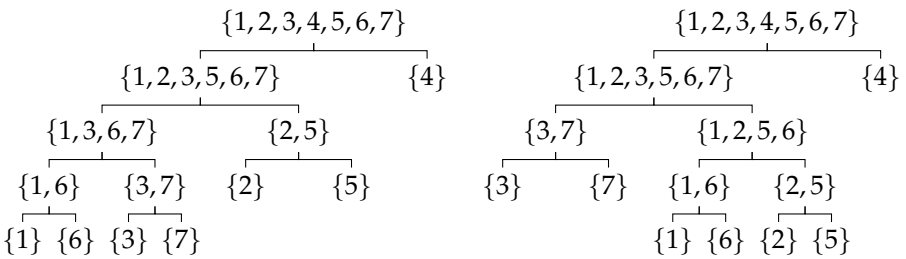

Figure 10
Two examples of recursive partitionings of a string of length 7.

Algorithm 2 Induction of a LCFRS from a string and a recursive partitioning.

Input: a string $w = \alpha_1 \cdots \alpha_n \in \Gamma^*$
a recursive partitioning π of w
Output: a LCFRS G that parses w according to π; fanout of G equals fanout of π

1: **function** INDUCE_LCFRS(w,π)
2: $P \leftarrow \emptyset$ ▷ set of LCFRS rules
3: **for** $p \in \mathrm{pos}(\pi)$ **do**
4: $j \leftarrow$ number of children of p in π
5: $J_0 \leftarrow \pi(p), J_1 \leftarrow \pi(p11), \ldots, J_j \leftarrow \pi(p1j)$
6: **if** $J_0 = \{i\}$ for some $i \in [n]$ **then**
7: $P \leftarrow P \cup \{(\!|\{i\}|\!)(\langle \alpha_i \rangle) \rightarrow \langle\rangle\}$
8: **else**
9: **for** $\ell \in [j]_0$ **do** $\langle J_{\ell,1}, \ldots, J_{\ell,k_\ell} \rangle \leftarrow \mathrm{spans}(J_\ell)$
10: **for** $\ell \in [j]$ **do** $m_\ell \leftarrow \sum_{j=1}^{\ell-1} k_j$
11: **for** $q \in [k_0]$ **do**
12: let r and $\ell_1, \ldots, \ell_r \in [j]$ and $q_1 \in [k_{\ell_1}], \ldots, q_r \in [k_{\ell_r}]$ be such that
$J_{0,q} = J_{\ell_1,q_1} \cup \cdots \cup J_{\ell_r,q_r}$ and $i \in J_{\ell_t,q_t}, i' \in J_{\ell_{t+1},q_{t+1}}$ implies $i < i'$
13: $s_q \leftarrow \langle x_{m_{\ell_1}+q_1}, \ldots, x_{m_{\ell_r}+q_r} \rangle$
14: $P \leftarrow P \cup \{(\!|J_0|\!)(s_{1,k_0}) \rightarrow \langle (\!|J_1|\!)(x_{1,k_1}), \ldots, (\!|J_j|\!)(x_{m_j+1,m_j+k_j}) \rangle\}$
15: **return** $G = (N, (\!|[n]|\!), \Gamma, P)$, where $N = \{(\!|J|\!) \mid J \text{ is label in } \pi\}$

label from π. (In practice, the nonterminals $(\!|J|\!)$ are replaced by other symbols, as will be explained in Section 6.8.) For each position p of π a rule is constructed. If p is a leaf labeled by $\{i\}$, then the constructed rule simply generates α_i where $(\!|\{i\}|\!)$ has fanout 1 (lines 6 and 7). For each internal position labeled J_0 and its children labeled $J_1, \ldots, J_j$ we compute the spans of these labels (line 9). For each argument q of $(\!|J_0|\!)$ a s-term s_q is constructed by analyzing the corresponding component $J_{0,q}$ of $\mathrm{spans}(J_0)$ (lines 12–13). Here we exploit the fact that $J_{0,q}$ can be decomposed in a unique way into a selection of sets that are among the spans of $J_1, \ldots, J_j$. Each of these sets translates to one variable. The resulting grammar G allows for exactly one derivation tree that derives w and has the same structure as π. We say that G **parses** w **according to** π.

We observe that G as constructed in this way is in a particular **normal form**.[3] To be precise, by the notation in Equation (2) each rule satisfies one of the following:

- $n \geq 2$ and $s_{1,k_0} \in T^*_\emptyset(X_{m_n})$, (**structural rule**)
- $n = 0$, $k_0 = 1$, and $s_1 = \langle \alpha \rangle$ for some $\alpha \in \Gamma$. (**terminal generating rule**)

Conversely, each derivation tree d of a LCFRS in normal form can be translated to a recursive partitioning π_d, by processing d bottom–up as follows. Each leaf, that is, a rule $A(\langle \alpha \rangle) \rightarrow \langle\rangle$, is replaced by the singleton set $\{i\}$ where i is the position of the corresponding occurrence of α in the accepted string. Each internal node is replaced by

3 See Seki et al. (1991) for an even stronger normal form.

the union of the sets that were computed for its children. For the constructed LCFRS G that parses w according to π and for its only derivation tree d, we have $\pi = \pi_d$.

Example 16
For a recursive partitioning of **h s g** in which the root has children labeled $\{1,3\}$ and $\{2\}$, Algorithm 2 constructs the LCFRS:

$$\begin{aligned}
&(\!|\{1,2,3\}|\!)(x_1x_3x_2) \to (\!|\{1,3\}|\!)(x_1,x_2)\ (\!|\{2\}|\!)(x_3)\\
&\quad (\!|\{1,3\}|\!)(x_1,x_2) \to (\!|\{1\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2)\\
&(\!|\{1\}|\!)(\mathbf{h}) \to \langle\rangle \quad (\!|\{2\}|\!)(\mathbf{s}) \to \langle\rangle \quad (\!|\{3\}|\!)(\mathbf{g}) \to \langle\rangle
\end{aligned}$$

Alternatively, if we let Algorithm 2 run on the recursive partitioning of **h s g** in which the root has children labeled $\{1,2\}$ and $\{3\}$, then it produces the following LCFRS, which is, in fact, a CFG:

$$\begin{aligned}
&(\!|\{1,2,3\}|\!)(x_1x_2) \to (\!|\{1,2\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2)\\
&\quad (\!|\{1,2\}|\!)(x_1x_2) \to (\!|\{1\}|\!)(x_1)\ (\!|\{2\}|\!)(x_2)\\
&(\!|\{1\}|\!)(\mathbf{h}) \to \langle\rangle \quad (\!|\{2\}|\!)(\mathbf{s}) \to \langle\rangle \quad (\!|\{3\}|\!)(\mathbf{g}) \to \langle\rangle
\end{aligned}$$

■

Observe that terminals are generated individually by the rules that were constructed from the leaves of the recursive partitioning but *not* by the structural rules obtained from internal nodes. Consequently, the LCFRS that we induce is in general not lexicalized. In order to obtain a lexicalized LCFRS, the notion of recursive partitioning would need to be generalized. We suspect that such a generalization is feasible, in conjunction with a corresponding generalization of the induction techniques presented in this article. This would be technically involved, however, and is therefore left for further research.

6.2 Construction of Recursive Partitionings

Figure 11 sketches three pipelines to induce a LCFRS from a hybrid tree, which differ in the way the recursive partitioning is constructed. The first way (cf. Figure 11a) is to extract a recursive partitioning directly from a hybrid tree $(s, \leq_s)$. This extraction is specified in Algorithm 3, which recursively traverses s. For each node in s, the gathered input positions consist of those obtained from the children (lines 6 and 7), plus possibly one input position from the node itself if its label is in Γ (line 5). The case distinction in line 8 and following is needed because every non-leaf in a recursive partitioning must have at least two children.

For a first example, consider the phrase structure in Figure 4a. The extracted recursive partitioning is given at the beginning of Example 16. For a second example, consider the dependency structure and the extracted recursive partitioning in Figure 12.

Note that if Algorithm 3 is applied to an arbitrary hybrid tree, then the accumulated set of input positions is empty for those positions of s that do not dominate elements with labels in Γ. The algorithm requires further (straightforward) refinements before it can be applied on hybrid trees with several roots, that is, if $|s| > 1$. In much of what follows, we ignore these special cases.

By this procedure, a recursive partitioning extracted from a discontinuous phrase structure or a non-projective dependency structure will have a fanout greater than 1. By then applying Algorithm 2, the resulting LCFRS will have fanout greater than 1. The more discontinuity exists in the input structures, the greater the fanout will be of the

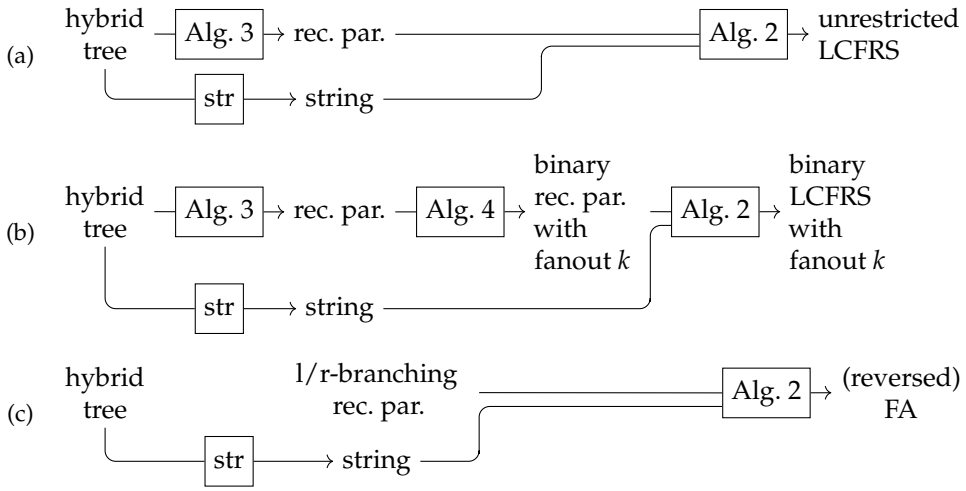

Figure 11
Three pipelines for LCFRS induction.

recursive partitioning and the resulting LCFRS, which in turn leads to greater parsing complexity. We therefore consider how to reduce the fanout of a recursive partitioning, before it is given as input to Algorithm 2. We aim to keep the structure of a given recursive partitioning largely unchanged, except where it exceeds a certain threshold on the fanout.

Algorithm 4 presents one possible such procedure. It starts at the root, which by definition has fanout 1. Assuming the fanout of the current node does not exceed k, then there are two cases to be distinguished. If the label J of the present node is a singleton,

Algorithm 3 Extraction of recursive partitioning from hybrid tree

Input: a hybrid tree $h = (s, \leq_s)$ with $\text{pos}_\Gamma(s) = \{p_1, \ldots, p_n\}$ where $p_i \leq_s p_{i+1}$ for each $i \in [n-1]$
Output: a recursive partitioning of $\text{str}(h)$

function EXTRACT_RECURSIVE_PARTITIONING(h)
  return REC_PAR(1)
function REC_PAR(p) ▷ $p \in \text{pos}(s)$
  $w \leftarrow \langle\rangle$ ▷ $w \in T^*_{\mathcal{P}([n])}$
  if $p = p_i \in \text{pos}_\Gamma(s)$ **then** $w \leftarrow \langle\{i\}\rangle$
  for $p' \in \text{children}(p)$ **do**
    $w \leftarrow w \cdot \text{REC\_PAR}(p')$
  if $|w| \leq 1$ **then return** w
  else
    $U \leftarrow \bigcup_{j=1}^{|w|} w(j)$
    sort w such that $\min(w(j)) < \min(w(j+1))$
    return $\langle U(w) \rangle$

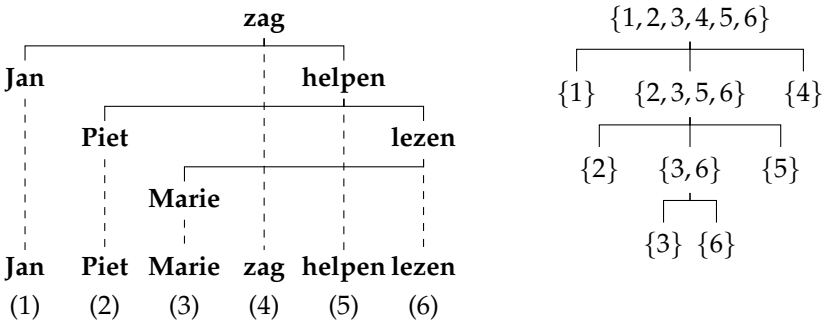

Figure 12
A dependency structure and the recursive partitioning extracted from it by Algorithm 3.

then the node is a leaf, and we can stop (lines 2–3). Otherwise, we search breadth-first through the subtree rooted in the present node to identify a descendant p such that both its label J' and $J \setminus J'$ have fanout not exceeding k (line 4). It is easy to see that such a node always exists: Ultimately, breadth-first search will reach the leaves, which are each labeled with a single number. One of these numbers must match either the lowest or the highest element of some maximal subset of consecutive numbers from J, so that the fanout of J cannot increase if that number is removed.

The current node is now given two children $\pi|_p$ and t. The first is the subtree rooted in the node labeled J' that we identified earlier, and the second is a copy of the subtree rooted in the present node, but with J' subtracted from the label of every node (lines 7–14). Nodes labeled with the empty set are removed (line 10), and if a node has the same label

Algorithm 4 Transformation of recursive partitioning

Input: a recursive partitioning π of a string of length n
an integer $k \geq 1$
Output: a binary recursive partitioning π' of fanout no greater than k

1: **function** TRANSFORM($\pi = \langle J(\langle t_1, \ldots, t_m \rangle) \rangle$)
2: **if** $|J| = 1$ **then**
3: **return** $\langle J(\langle\rangle) \rangle$
4: breadth-first search p in $\text{pos}(\pi) \setminus \{1\}$ such that $\pi(p)$ and $J \setminus \pi(p)$ have fanout $\leq k$
5: $t \leftarrow$ FILTER($\pi(p), \pi$)
6: **return** $\langle J(\text{TRANSFORM}(\langle \pi|_p \rangle), \text{TRANSFORM}(t)) \rangle$
7: **function** FILTER($J', \pi = \langle J(\langle t_1, \ldots, t_m \rangle) \rangle$) ▷ $J' \subseteq [n], \pi \in T^*_{\mathcal{P}([n])}$
8: $F \leftarrow J \setminus J'$
9: **if** $|F| = 1$ **then** **return** $\langle F(\langle\rangle) \rangle$
10: **else if** $|F| = 0$ **then return** $\langle\rangle$
11: **else**
12: $s \leftarrow$ FILTER($J', \langle t_1 \rangle$) $\cdot \ldots \cdot$ FILTER($J', \langle t_m \rangle$)
13: **if** $|s| = 1$ **then** **return** s
14: **else** **return** $\langle F(s) \rangle$

as its parent then the two are collapsed (line 13). As the two children each have fanout not exceeding k, we can apply the procedure recursively (line 6).

Example 17
The recursive partitioning π in the left half of Figure 10 has a node labeled $\{1,3,6,7\}$, with fanout 3. With $J = \{1,2,3,5,6,7\}$ and $k = 2$, one possible choice for J' is $\{3,7\}$, as then both J' and $J \setminus J' = \{1,2,5,6\}$ have fanout not exceeding 2. This leads to the partitioning π' in the right half of the figure. Because now all node labels have fanout not exceeding 2, recursive traversal will make no further changes. The partitioning π' is similar to π in the sense that subtrees that are not on the path to $\{3,7\}$ remain unchanged. Other valid choices for J' would be $\{2\}$ and $\{5\}$. Not a valid choice for J' would be $\{1,6\}$, as $J \setminus \{1,6\} = \{2,3,5,7\}$, which has fanout 3. ■

Algorithm 4 ensures that subsequent induction of an LCFRS (cf. Figure 11b) leads to a binary LCFRS. Note the difference between binarization algorithms such as those from Gómez-Rodríguez and Satta (2009) and Gómez-Rodríguez et al. (2009), which are applied on grammar rules, and our procedure, which is applied *before* any grammar is obtained. Unlike van Cranenburgh (2012), moreover, our objective is not to obtain a "coarse" grammar for the purpose of coarse-to-fine parsing.

Note that if k is chosen to be 1, then the resulting partitioning is consistent with derivations of a CFG. Even simpler partitionings exist. In particular, the **left-branching** partitioning has internal node labels that are $\{1,2,\ldots,m\}$, each with children labeled $\{1,\ldots,m-1\}$ and $\{m\}$. These are consistent with the computations of finite automata (FA) in reverse direction; see Figure 11c. Similarly, there is a **right-branching** recursive partitioning, reflecting finite-state processing in a forward direction. The use of branching partitionings completely detaches string parsing from the structure of the given hybrid tree.

Example 18
The right-branching recursive partitioning for the dependency structure in Figure 12 and the transitions of the FA obtained from it are depicted in Figure 13. The FA can be obtained via a LCFRS that is induced from the recursive partitioning, as usual. By construction it is equivalent to a right-linear CFG, with rules:

$$(\!|\{i, i+1\ldots,6\}|\!)(x_1x_2) \to \langle (\!|\{i\}|\!)(x_1), (\!|\{i+1,\ldots,6\}|\!)(x_2)\rangle$$

for each $i \in [5]$, and $(\!|\{i\}|\!)(\alpha_i) \to \langle\rangle$ for each $i \in [6]$. ■

The relation between recursive partitioning and worst-case parsing complexity of the induced LCFRS is summarized in Table 1. We remark that the parsing complexity for

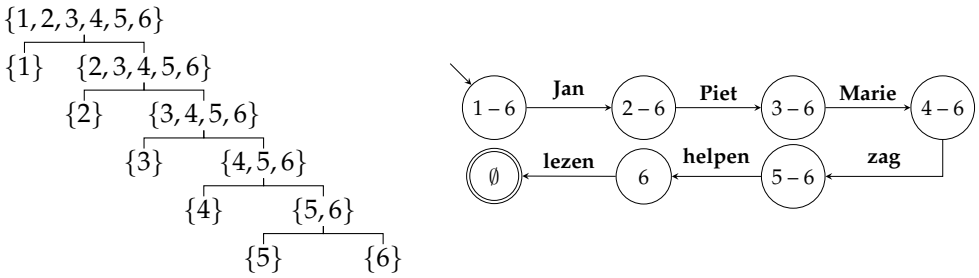

Figure 13
A right-branching recursive partitioning and an FA.

Table 1
Induction pipeline (from Figure 11) and type and worst-case parsing complexity of the induced LCFRS. Here k and m denote the fanout and the maximum length of any right-hand side of the LCFRS, respectively.

Pipeline	Type of the induced LCFRS	Parsing complexity for string of length n
(a)	LCFRS of arbitrary k and m	$\mathcal{O}(n^{(m+1)\cdot k})$
(b) $k \geq 1$	binarized k-LCFRS	$\mathcal{O}(n^{3k})$
(b) $k = 1$	binarized CFG	$\mathcal{O}(n^3)$
(c) right	FA	$\mathcal{O}(n)$
(c) left	(reverse) FA	$\mathcal{O}(n)$

unrestricted LCFRSs would improve from $\mathcal{O}(n^{(m+1)\cdot k})$ to $\mathcal{O}(n^{2\cdot k+2})$ if we could ensure that the LCFRSs are well-nested (Gómez-Rodríguez, Kuhlmann, and Satta 2010), or in other words, if we replace LCFRSs by sMGs. How to refine pipeline (a) to achieve this is left for future investigation.

6.3 Induction of Hybrid Grammars

In the remainder of Section 6 we extend the induction pipelines for LCFRS to induction pipelines for LCFRS/sDCP hybrid grammars, as illustrated in Figure 14. Given a hybrid tree $h = (s, \leq_s)$, we choose a recursive partitioning π obtained in one of the ways discussed in the previous section. We apply Algorithm 2 to induce a LCFRS G_1 that generates $\text{str}(h)$. Using the same recursive partitioning π, we induce a sDCP G_2 that generates s. For this we use either Algorithm 5 or 6, depending on whether h is a phrase structure or a dependency structure.

Then we combine both grammars into a hybrid grammar G, which synchronously generates the hybrid tree h. For this, we synchronize terminals and nonterminals of G_1 and G_2 (via indexed symbols) by slightly changing Algorithm 2. Because the terminals that require synchronization are only generated by the rules constructed for the leaves of π, we alter line 7 to

$$P \leftarrow P \cup \{(\!|\{i\}|\!)(\alpha_i^{\boxed{i}}) \rightarrow \langle\rangle\}$$

Likewise, we need to index the nonterminals on the right-hand side of each structural rule, that is, we alter line 14 to

$$P \leftarrow P \cup \{(\!|J_0|\!)(s_{1,k_0}) \rightarrow \langle (\!|J_1|\!)^{\boxed{1}}(x_{1,k_1}), \ldots, (\!|J_j|\!)^{\boxed{j}}(x_{m_j+1,m_j+k_j})\rangle\}$$

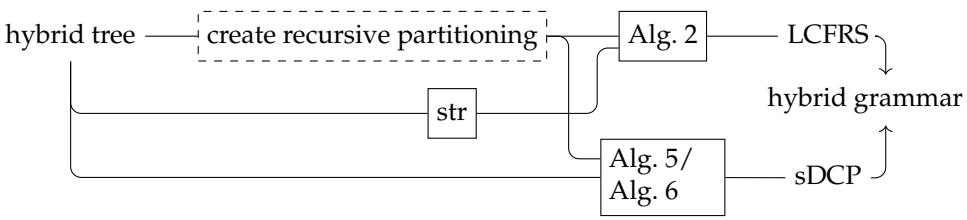

Figure 14
Pipeline for the induction of LCFRS/sDCP hybrid grammars.

The corresponding indexing needed in G_2 will be described in Sections 6.4 and 6.6.

In preparation, we define several notions to relate labels of the recursive partitioning with subsets of the positions of s. Let $\text{pos}_\Gamma(s) = \{p_1, \ldots, p_n\}$ with $p_i \leq_s p_{i+1}$ $(i \in [n-1])$ and let J be a label of π. We define the set $\Pi(J) = \{p_i \mid i \in J\}$, which identifies the nodes of s corresponding to the elements in J. For any subset $U \subseteq \text{pos}(s)$, we construct the sets $\top(U)$ and $\bot(U)$, which, intuitively, delimit U from the top and the bottom, respectively. Formally, $p \in \top(U)$ if and only if $p \in U$ and $\text{parent}(p) \notin U$. We have $p \in \bot(U)$ if and only if $\text{parent}(p) \in U$ and $p \notin U$.

6.4 Induction of LCFRS/sDCP Hybrid Grammars from Phrase Structures

Grammar induction is relatively straightforward for a given hybrid tree $h = (s, \leq_s)$ that is a phrase structure, and a given recursive partitioning π. For each node of π, its label is a set of positions of $\text{str}(h)$, and these positions must each correspond to a leaf node of s, by virtue of h being a phrase structure. We can apply a closure operation on this set of leaf nodes that includes a node if all of its children are included. Formally, let J be a label of π. Then $C(J)$ is the smallest set $U \subseteq \text{pos}(s)$ satisfying (i) $\Pi(J) \subseteq U$ and (ii) if $p \in \text{pos}(s)$, $\text{children}(p) \neq \emptyset$, and $\text{children}(p) \subseteq U$, then $p \in U$.

The set $C(J)$ corresponds to a set of (maximal, disjoint) sub-s-terms of s, that is, $C(J)$ can be partitioned such that each part contains the positions that correspond to one sub-s-term. These parts can be arranged according to the lexicographical ordering on positions in s. Formally, for each set $U \subseteq \text{pos}(s)$, we define $\text{s-rk}(U)$ to be the maximal number k such that U can be partitioned into sets $U_1, \ldots, U_k$ where:

- for every $i \in [k]$, $p \in U_i$, and $p' \in U$ if $p' = \text{parent}(p)$ or $p' = \text{right-sibling}(p)$, then $p' \in U_i$, and
- for every $i, j \in [k]$, $p \in U_i$, and $p' \in U_j$ we have $p \leq_\ell p'$ implies $i \leq j$.

Note that $U_1, \ldots, U_k$ are uniquely defined by this, and we write $\text{gspans}(U) = \langle U_1, \ldots, U_k \rangle$. The function "gspans" generalizes "spans," which we defined for recursive partitioning earlier. Similarly, s-rk as used here generalizes the notion of fanout as used for node labels of a recursive partitioning.

These concepts allow us to define a mapping from a node J in π to the set $\text{gspans}(C(J))$ of positions of s. This mapping is such that if $J_1, \ldots, J_m$ are the child nodes of node J in π, then each set $U_j^{(i)}$ in $\text{gspans}(C(J_i))$ $(i \in [m], j \in [\text{s-rk}(J_i)])$ is contained in some set $U_q^{(0)}$ in $\text{gspans}(C(J))$ $(q \in [\text{s-rk}(J)])$. A different way of looking at this is that the image of J can be constructed out of the images of J_i $(i \in [m])$, possibly by adding further nodes linking existing sub-s-terms together.

Such composition of s-terms into larger s-terms is realized by a sDCP without inherited arguments, as constructed by Algorithm 5. It builds one production for each node J_0 with children $J_1, \ldots, J_m$. J_0 has a synthesized attribute for each set $U_q^{(0)}$ in $\text{gspans}(C(J_0))$. The corresponding s-term s_q is constructed by traversing the positions in $U_q^{(0)}$ (lines 18–31). A sequence of consecutive positions that are also in some $U_j^{(i)}$ in $\text{gspans}(C(J_i))$ is realized by a single variable $x_j^{(i)}$ (lines 21–24). Remaining positions become nodes of s_q (lines 26–30).

In order to create a hybrid rule as outlined in Section 6.3, we need to introduce indexing of nonterminals and terminals in the sDCP. Synchronized terminals are only

Algorithm 5 Induction of a sDCP from a phrase structure and a recursive partitioning

Input: a phrase structure $h = (s, \leq_s)$ with $\mathrm{pos}_\Gamma(s) = \{p_1, \ldots, p_n\}$ where $p_i <_s p_{i+1}$ for each $i \in [n-1]$
a recursive partitioning π of $\mathrm{str}(s)$
Output: a sDCP G that generates s according to π

function CONSTRUCT_SDCP($(s, \leq_s)$,π)
  $P \leftarrow \emptyset$ ▷ set of sDCP rules
  for $p \in \mathrm{pos}(\pi)$ **do**
    $m \leftarrow$ number of children of p in π
    $J_0 \leftarrow \pi(p), J_1 \leftarrow \pi(p11), \ldots, J_m \leftarrow \pi(p1m)$
    if $J_0 = \{i\}$ for some $i \in [n]$ **then**
      $\langle U_1^{(0)} \rangle \leftarrow \mathrm{gspans}(C(J_0))$
      $\{p\} \leftarrow \top(U_1^{(0)})$
      $P \leftarrow P \cup \{(\!|J_0|\!)(\langle s|_p \rangle) \to \langle\rangle\}$
    else
      $\langle U_1^{(0)}, \ldots, U_{k_0'}^{(0)} \rangle \leftarrow \mathrm{gspans}(C(J_0))$
      $\langle U_1^{(i)}, \ldots, U_{k_i}^{(i)} \rangle \leftarrow \mathrm{gspans}(C(J_i))$ for each $i \in [m]$
      for $q \in [k_0']$ **do**
        $p_{\min} \leftarrow \min_{\leq_\ell}(U_q^{(0)})$
        $s_q \leftarrow$ CONSTRSTERM($p_{\min}, U_q^{(0)}, (U_j^{(i)} \mid i \in [m], j \in [k_i])$)
      $P \leftarrow P \cup \{(\!|J_0|\!)(s_{1,k_0'}) \to \langle (\!|J_1|\!)(x_{1,k_1}^{(1)}), \ldots, (\!|J_m|\!)(x_{1,k_m}^{(m)}) \rangle\}$
  return $G = (N, (\!|[n]|\!), \Sigma, P)$, where $N = \{(\!|J|\!) \mid J \text{ label in } \pi\}$

function CONSTRSTERM($p, U_q^{(0)}, (U_j^{(i)} \mid i \in [m], j \in [k_i])$)
  ▷ $p \in \mathrm{pos}(s)$, $U_q^{(0)} \subseteq \mathrm{pos}(s)$, $(U_j^{(i)} \mid i \in [m], j \in [k_i])$ family of subsets of $\mathrm{pos}(s)$
  $s' \leftarrow \varepsilon$
  while $p \in U_q^{(0)}$ **do**
    if $p \in U_j^{(i)}$ for some $i \in [m], j \in k_i$ **then**
      $s' \leftarrow s' \cdot \langle x_j^{(i)} \rangle$
      while $p \in U_j^{(i)}$ **do**
        $p \leftarrow$ right-sibling(p)
    else
      $p' \leftarrow \min_{\leq_\ell}(\mathrm{children}(p))$
      $c \leftarrow$ CONSTRSTERM($p', U_q^{(0)}, (U_j^{(i)} \mid i \in [m], j \in [k_i])$)
      $\sigma \leftarrow s(p)$
      $s' \leftarrow s' \cdot \langle \sigma(c) \rangle$
      $p \leftarrow$ right-sibling(p)
  return s'

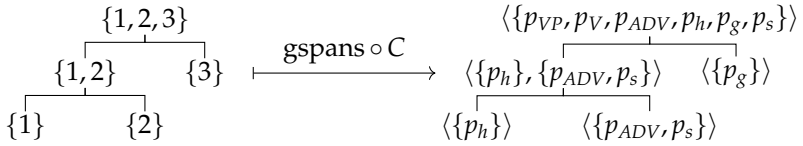

Figure 15
The left-branching recursive partitioning of $\{1,2,3\}$ and the image under gspans after C.

generated by the rules constructed for the leaves of π. Hence, we alter line 9 in Algorithm 5 to:

$$P \leftarrow P \cup \{(\!|J_0|\!)(\langle s[s(p_i)^{\boxed{1}}]_{p_i}|_p\rangle) \rightarrow \langle\rangle\}$$

that is, we add an index to the symbol at position p_i before selecting the relevant subtree $s|_p$ of s. For the indexing of nonterminals, we change line 16 to:

$$P \leftarrow P \cup \{(\!|J_0|\!)(s_{1,k_0'}) \rightarrow \langle (\!|J_1|\!)^{\boxed{1}}(x^{(1)}_{1,k_1}), \ldots, (\!|J_m|\!)^{\boxed{m}}(x^{(m)}_{1,k_m})\rangle\}$$

Example 19
Consider the hybrid tree h in Figure 4a, in combination with the recursive partitioning extracted from the hybrid tree by Algorithm 3. The children of the root are $\{1,3\}$ and $\{2\}$. The relevant s-term for $\{1,3\}$ is $\mathbf{V}(\mathbf{hat}, \mathbf{gearbeitet})$ and the s-term for $\{2\}$ is $\mathbf{ADV}(\mathbf{schnell})$. Application of Algorithm 5 yields the sDCP grammar:

$$(\!|\{1,2,3\}|\!)(\mathbf{VP}(x_1x_2)) \rightarrow (\!|\{1,3\}|\!)(x_1)\ (\!|\{2\}|\!)(x_2)$$
$$(\!|\{1,3\}|\!)(\mathbf{V}(x_1x_2)) \rightarrow (\!|\{1\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2)$$
$$(\!|\{1\}|\!)(\mathbf{h}) \rightarrow \langle\rangle \quad (\!|\{2\}|\!)(\mathbf{ADV}(\mathbf{s})) \rightarrow \langle\rangle \quad (\!|\{3\}|\!)(\mathbf{g}) \rightarrow \langle\rangle$$

Synchronizing this sDCP with the first LCFRS of Example 16 yields the following LCFRS/sDCP hybrid grammar:

$$[(\!|\{1,2,3\}|\!)(x_1x_3x_2) \rightarrow (\!|\{1,3\}|\!)^{\boxed{1}}(x_1,x_2)\ (\!|\{2\}|\!)^{\boxed{2}}(x_3),$$
$$(\!|\{1,2,3\}|\!)(\mathbf{VP}(x_1x_2)) \rightarrow (\!|\{1,3\}|\!)^{\boxed{1}}(x_1)\ (\!|\{2\}|\!)^{\boxed{2}}(x_2)]$$

$$[(\!|\{1,3\}|\!)(x_1,x_2) \rightarrow (\!|\{1\}|\!)^{\boxed{1}}(x_1)\ (\!|\{3\}|\!)^{\boxed{2}}(x_2),$$
$$(\!|\{1,3\}|\!)(\mathbf{V}(x_1x_2)) \rightarrow (\!|\{1\}|\!)^{\boxed{1}}(x_1)\ (\!|\{3\}|\!)^{\boxed{2}}(x_2)]$$

$$[(\!|\{1\}|\!)(\mathbf{h}^{\boxed{1}}) \rightarrow \langle\rangle\ ,\ (\!|\{1\}|\!)(\mathbf{h}^{\boxed{1}}) \rightarrow \langle\rangle]$$

$$[(\!|\{2\}|\!)(\mathbf{s}^{\boxed{1}}) \rightarrow \langle\rangle\ ,\ (\!|\{2\}|\!)(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \rightarrow \langle\rangle]$$

$$[(\!|\{3\}|\!)(\mathbf{g}^{\boxed{1}}) \rightarrow \langle\rangle\ ,\ (\!|\{3\}|\!)(\mathbf{g}^{\boxed{1}}) \rightarrow \langle\rangle]$$

The fanout of the LCFRS is 2 and the s-rank of the sDCP is 1. ■

Example 20
Consider again the hybrid tree in Figure 4a, but now in combination with the recursive partitioning shown in Figure 15, that is, the left-branching partitioning of $\{1,2,3\}$. The relevant disjoint subtrees for $\{1,2\}$ are **hat** and **ADV**(**schnell**) and the one for $\{3\}$ is **gearbeitet**. (In a real-world grammar we would have parts of speech occurring above all

the words.) Applying Algorithm 5 and synchronizing the resulting sDCP grammar with the second LCFRS of Example 16 yields the following LCFRS/sDCP hybrid grammar:

$$[(\!|\{1,2,3\}|\!)(x_1x_2) \to (\!|\{1,2\}|\!)^{\boxed{1}}(x_1)\ (\!|\{3\}|\!)^{\boxed{2}}(x_2)\,,$$
$$\quad (\!|\{1,2,3\}|\!)(\mathbf{VP}(\mathbf{V}(x_1x_3)x_2)) \to (\!|\{1,2\}|\!)^{\boxed{1}}(x_1,x_2)\ (\!|\{3\}|\!)^{\boxed{2}}(x_3)]$$

$$[(\!|\{1,2\}|\!)(x_1x_2) \to (\!|\{1\}|\!)^{\boxed{1}}(x_1)\ (\!|\{2\}|\!)^{\boxed{2}}(x_2)\,,$$
$$\quad (\!|\{1,2\}|\!)(x_1,x_2)) \to (\!|\{1\}|\!)^{\boxed{1}}(x_1)\ (\!|\{2\}|\!)^{\boxed{2}}(x_2)]$$

$$[(\!|\{1\}|\!)(\mathbf{h}^{\boxed{1}}) \to \langle\rangle\ ,\ (\!|\{1\}|\!)(\mathbf{h}^{\boxed{1}}) \to \langle\rangle]$$

$$[(\!|\{2\}|\!)(\mathbf{s}^{\boxed{1}}) \to \langle\rangle\ ,\ (\!|\{2\}|\!)(\mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \to \langle\rangle]$$

$$[(\!|\{3\}|\!)(\mathbf{g}^{\boxed{1}}) \to \langle\rangle\ ,\ (\!|\{3\}|\!)(\mathbf{g}^{\boxed{1}}) \to \langle\rangle]$$

The fanout of the LCFRS is 1 and the s-rank of the sDCP is 2. ■

Figure 16 compares the two derivation trees of the LCFRS/sDCP hybrid grammars of Examples 19 (left) and 20 (right).

We observe the following general property of Algorithm 5: If the recursive partitioning extracted by Algorithm 3 is given as input, then each nonterminal of the induced sDCP G has s-rank 1 (as in Example 19). This coincides with pipeline (a) of Figure 11, that is, the induction of a LCFRS of arbitrary fanout. However, if the recursive partitioning is transformed by Algorithm 4 or if the left-branching or right-branching recursive partitioning is used (as in Example 20 and as in pipelines (b) and (c) of Figure 11), then the fanout of the induced LCFRS decreases and its derivations are binarized. At the same time, the numbers of synthesized arguments in the induced sDCP may increase. In other words, we witness a trade-off between the degree of mild context-sensitivity of the LCFRS and the numbers of arguments of the sDCP.

We conclude:

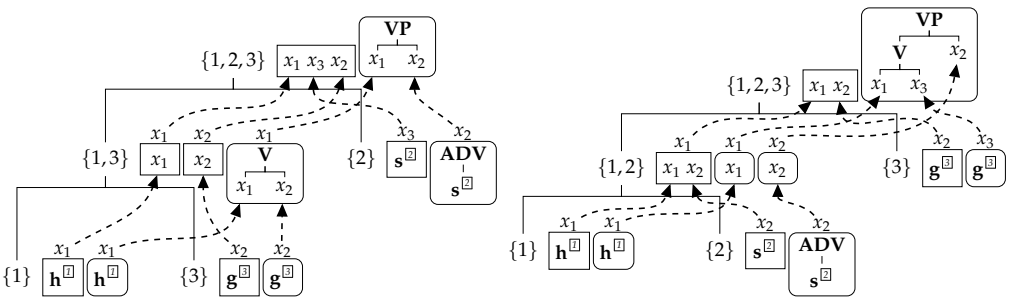

Figure 16
(Left) The only derivation tree of the LCFRS/sDCP hybrid grammar induced in Example 19 with the recursive partitioning extracted from the hybrid tree. (Right) The only derivation tree of the LCFRS/sDCP hybrid grammar induced in Example 20 with the left-branching recursive partitioning.

Theorem 3
For each phrase structure h and recursive partitioning π of str(h), we can construct a LCFRS/sDCP hybrid grammar G such that G generates h and parses str(h) according to π. Moreover, the sDCP that is the second component only has synthesized arguments.

6.5 Induction of LCFRS/sCFTG Hybrid Grammars from Phrase Structures

Given a recursive partitioning π and a phrase structure $h = (s, \leq_s)$, the construction in Section 6.4 relied on a mapping from a node of π labeled J to sets of positions of maximal sub-s-terms in s whose yields together cover exactly the positions in J. We now say π is **chunky** with respect to h if, for each node J of π, we have s-rk($C(J)$) $= 1$, that is, the nodes in its image under the mapping form a single sub-s-term. If π is chunky with respect to h, then each sDCP nonterminal in the construction from Section 6.4 will have a single synthesized argument. A sDCP with this property is equivalent to a sCFTG in which all nonterminals have rank 0, that is, a regular tree grammar. Therefore:

Theorem 4
For each phrase structure h and recursive partitioning π of str(h) that is chunky with respect to h, we can construct a LCFRS/sCFTG hybrid grammar G such that G generates h and parses str(h) according to π. Moreover, the second component of G is a regular tree grammar.

6.6 Induction of LCFRS/sDCP Hybrid Grammars from Dependency Structures

Let $h = (s, \leq_s)$ be a hybrid tree over (Σ, Σ), or in other words, a dependency structure, and let π be a recursive partitioning of str(h). The task is to construct a LCFRS/sDCP hybrid grammar that generates h and parses str(h) according to π.

In the case of phrase structures, we could identify entire subtrees of s whose yields corresponded to subsets of node labels of π. A sequence of consecutive such subtrees (i.e., whose roots were siblings) was then translated to a synthesized argument of a sDCP rule. With dependency structures, however, we need to allow for the scenario where we have a label J of a node in π and two positions p and p' in s with parent(p') $= p$, and $p \in \Pi(J)$ whereas $p' \notin \Pi(J)$. Thus we will need to consider a subtree of s rooted in p, with a "gap" for child p'. This gap will be implemented by introducing an inherited argument in the sDCP, so that the gap can be filled by a structure built elsewhere.

Like the induction algorithms considered before, Algorithm 6 constructs a rule for each node p of π. If p is a leaf of π (line 6), labeled with a singleton $J_0 = \{i\}$, we construct the sDCP rule $(\!|J_0|\!)(x_1, \langle\alpha(\langle x_1\rangle)\rangle) \to \langle\rangle$, where α is the i-th element of str(h), assuming the node labeled α is not a leaf in s (line 10). The i-rank of $(\!|J_0|\!)$ is 1. Coupled to the corresponding LCFRS rule, this creates the hybrid rule $[(\!|J_0|\!)(\langle\alpha^{\boxed{1}}\rangle) \to \langle\rangle, (\!|J_0|\!)(x_1, \langle\alpha^{\boxed{1}}(\langle x_1\rangle)\rangle) \to \langle\rangle]$. If the node labeled α is a leaf of s, we can dispense with the inherited argument of $(\!|J_0|\!)$ in the second component and replace $\alpha(\langle x_1\rangle)$ by α (line 8).

If p is an internal node of π, then we proceed as follows. We determine the set $\Pi(J_0)$ of positions of s that correspond to the numbers in the label of p. We compute the sets of positions $\top(\Pi(J_0))$ and $\bot(\Pi(J_0))$ that delimit $\Pi(J_0)$ from the top and bottom, respectively. We bundle consecutive positions in $\top(\Pi(J_0))$ and $\bot(\Pi(J_0))$ by applying gspans. For brevity, we write $\top_{\max}(J_0)$ instead of gspans($\top(\Pi(J_0))$) and $\bot_{\max}(J_0)$ instead of gspans($\bot(\Pi(J_0))$). The nonterminal $(\!|J_0|\!)$ is given a synthesized attribute for each set $I_q^{(0)}$ in $\top_{\max}(J_0)$ (line 12), and an inherited attribute for each set $O_j^{(0)}$ in $\bot_{\max}(J_0)$ (line 13).

Algorithm 6 Induction of a sDCP from a dependency structure and a recursive partitioning

Input: a dependency structure $h = (s, \leq_s)$ with $\mathrm{pos}(s) = \{p_1, \ldots, p_n\}$ where $p_i <_s p_{i+1}$ for each $i \in [n-1]$
a recursive partitioning π of $\mathrm{str}(s)$
Output: a sDCP G that generates s according to π

1: **function** CONSTRUCT_SDCP($(s, \leq_s)$,π)
2: $P \leftarrow \emptyset$ ▷ set of sDCP rules
3: **for** $p \in \mathrm{pos}(\pi)$ **do**
4: $m \leftarrow$ number of children of p in π
5: $J_0 \leftarrow \pi(p), J_1 \leftarrow \pi(p11), \ldots, J_m \leftarrow \pi(p1m)$
6: **if** $J_0 = \{i\}$ for some $i \in [n]$ and $s(p_i) = \alpha$ **then**
7: **if** $\mathrm{children}(p_i) = \emptyset$ **then**
8: $R \leftarrow R \cup \{(\!|J_0|\!)(\langle\alpha\rangle) \to \langle\rangle\}$
9: **else**
10: $R \leftarrow R \cup \{(\!|J_0|\!)(x_1, \langle\alpha(\langle x_1\rangle)\rangle) \to \langle\rangle\}$
11: **else**
12: $\langle I_1^{(0)}, \ldots, I_{k_0'}^{(0)}\rangle \leftarrow \top_{\max}(J_0)$
13: $\langle O_1^{(0)}, \ldots, O_{k_0}^{(0)}\rangle \leftarrow \bot_{\max}(J_0)$
14: $\langle O_1^{(i)}, \ldots, O_{k_i}^{(i)}\rangle \leftarrow \top_{\max}(J_i)$ for each $i \in [m]$
15: $\langle I_1^{(\ell)}, \ldots, I_{k_i'}^{(\ell)}\rangle \leftarrow \bot_{\max}(J_\ell)$ for each $\ell \in [m]$
16: **for** each $\ell \in [m]_0$ and $q \in [k_\ell']$ **do**
17: $s_q^{(\ell)} \leftarrow$ CONSTRSTERM($I_q^{(\ell)}$, $(O_j^{(i)} \mid i \in [m]_0, j \in [k_i])$)
18: $P \leftarrow P \cup \{(\!|J_0|\!)(x_{1,k_0}^{(0)}, s_{1,k_0'}^{(0)}) \to \langle (\!|J_1|\!)(s_{1,k_1'}^{(1)}, x_{1,k_1}^{(1)}), \ldots, (\!|J_m|\!)(s_{1,k_m'}^{(m)}, x_{1,k_m}^{(m)})\rangle\}$
19: **return** $G = (N, (\!|[n]|\!), \Sigma, P)$, where $N = \{(\!|J|\!) \mid J \text{ label in } \pi\}$

20: **function** CONSTRSTERM(I, $(O_j^{(i)} \mid i \in [m]_0, j \in [k_i])$)
▷ $I \subseteq \mathrm{pos}(s)$, $(O_j^{(i)} \mid i \in [m]_0, j \in [k_i])$ family of subsets of $\mathrm{pos}(s)$
21: $s' \leftarrow \varepsilon$
22: **while** $I \neq \emptyset$ **do**
23: $p \leftarrow \min_{\leq_\ell}(I)$
24: let i in $[m]_0$ and j in $[k_i]$ such that $p \in O_j^{(i)}$
25: $I \leftarrow I \setminus O_j^{(i)}$
26: $s' \leftarrow s' \cdot \langle x_j^{(i)}\rangle$
27: **return** s'

Analogously, we determine sets $O_j^{(i)}$ and $I_q^{(\ell)}$ of positions of s for each child of p labeled J_i (lines 14–15). Each set $O_j^{(i)}$ corresponds to a distinct variable $x_j^{(i)}$ in the sDCP rule. Each set $I_q^{(\ell)}$ corresponds to a s-term $s_q^{(\ell)}$ which combines variables. In particular, each set $I_q^{(\ell)}$ is the (disjoint) union of some of the sets $O_j^{(i)}$. Conversely, each set $O_j^{(i)}$ is disjoint with all but one of the sets $I_q^{(\ell)}$. This fact is used in lines 20–27 to construct the

s-term $s_q^{(\ell)}$. Having specified the variables and s-terms, the construction of the sDCP rule is completed in line 18. Much as in Section 6.4, this sDCP rule can be coupled to the corresponding LCFRS rule to form a hybrid rule.

Example 21
Figure 17 presents part of the hybrid tree from Figure 4b, together with part of a recursive partitioning. Let us use the symbols p_P, p_M, p_h, p_l for the four positions in the hybrid tree corresponding to the string positions $2, 3, 5, 6$. Naturally, $\Pi(\{2,3,5,6\}) = \{p_P, p_M, p_h, p_l\}$, $\Pi(\{2,6\}) = \{p_P, p_l\}$, $\Pi(\{3,5\}) = \{p_M, p_h\}$.

If we investigate which nodes in the hybrid tree delimit these sets from the top and from the bottom, we obtain:

$$\top(\Pi(\{2,3,5,6\})) = \{p_h\} \quad \top(\Pi(\{2,6\})) = \{p_P, p_l\} \quad \top(\Pi(\{3,5\})) = \{p_M, p_h\}$$
$$\bot(\Pi(\{2,3,5,6\})) = \emptyset \quad \bot(\Pi(\{2,6\})) = \{p_M\} \quad \bot(\Pi(\{3,5\})) = \{p_P, p_l\}$$

We obtain maximal sets of consecutive positions by:

$$\top_{\max}(\{2,3,5,6\}) = \langle\{p_h\}\rangle \quad \top_{\max}(\{2,6\}) = \langle\{p_P, p_l\}\rangle \quad \top_{\max}(\{3,5\}) = \langle\{p_M\}, \{p_h\}\rangle$$
$$\bot_{\max}(\{2,3,5,6\}) = \langle\rangle \quad \bot_{\max}(\{2,6\}) = \langle\{p_M\}\rangle \quad \bot_{\max}(\{3,5\}) = \langle\{p_P, p_l\}\rangle$$

One constructed hybrid rule is therefore:

$$[(\!|\{2,3,5,6\}|\!)(x_1x_3, x_4x_2) \to (\!|\{2,6\}|\!)^{\boxed{1}}(x_1, x_2)\ (\!|\{3,5\}|\!)^{\boxed{2}}(x_3, x_4),$$
$$(\!|\{2,3,5,6\}|\!)(x_3) \to (\!|\{2,6\}|\!)^{\boxed{1}}(x_2, x_1)\ (\!|\{3,5\}|\!)^{\boxed{2}}(x_1, x_2, x_3)]$$

The first component was constructed as in previous sections. In the second component, the i-rank of $(\!|\{2,3,5,6\}|\!)$ is 0 as $\bot_{\max}(\{2,3,5,6\})$ has length 0. The i-rank of $(\!|\{3,5\}|\!)$ is 1 as $\bot_{\max}(\{3,5\})$ contains a single set $\{p_P, p_l\}$, whereas its s-rank is 2 as $\top_{\max}(\{3,5\})$ contains two sets. Because the first (and only) set of $\bot_{\max}(\{2,6\})$, namely $\{p_M\}$, equals the first set of $\top_{\max}(\{3,5\})$, the variable x_2 is shared between the inherited argument of $(\!|\{2,6\}|\!)$ and the first synthesized argument of $(\!|\{3,5\}|\!)$.

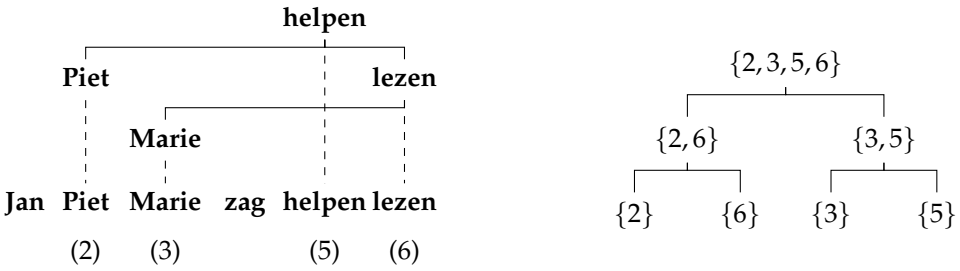

Figure 17
Part of a hybrid tree and part of a recursive partitioning.

Some other rules in this example are:

$$[(\!|\{2,6\}|\!)(x_1,x_2) \rightarrow (\!|\{2\}|\!)^{\boxed{1}}(x_1)\ (\!|\{6\}|\!)^{\boxed{2}}(x_2)\ ,\ (\!|\{2,6\}|\!)(x_1,x_2x_3) \rightarrow (\!|\{2\}|\!)^{\boxed{1}}(x_2)\ (\!|\{6\}|\!)^{\boxed{2}}(x_1,x_3)]$$

$$[(\!|\{3,5\}|\!)(x_1,x_2) \rightarrow (\!|\{3\}|\!)^{\boxed{1}}(x_1)\ (\!|\{5\}|\!)^{\boxed{2}}(x_2)\ ,\ (\!|\{3,5\}|\!)(x_1,x_2,x_3) \rightarrow (\!|\{3\}|\!)^{\boxed{1}}(x_2)\ (\!|\{5\}|\!)^{\boxed{2}}(x_1,x_3)]$$

$$[(\!|\{5\}|\!)(\mathbf{helpen}^{\boxed{1}}) \rightarrow \langle\rangle\ ,\ (\!|\{5\}|\!)(x_1,\mathbf{helpen}^{\boxed{1}}(x_1)) \rightarrow \langle\rangle]$$

For the second component of the first hybrid rule, we remark that $\bot_{\max}(\{6\}) = \langle\{p_M\}\rangle$ and therefore $(\!|\{6\}|\!)$ has i-rank 1. Further, $\top_{\max}(\{2\}) = \langle\{p_P\}\rangle$, $\top_{\max}(\{6\}) = \langle\{p_l\}\rangle$, and we already saw that $\top_{\max}(\{2,6\}) = \langle\{p_P,p_l\}\rangle$. The fact that $\{p_P,p_l\} = \{p_P\} \cup \{p_l\}$ and $p_P <_\ell p_l$ explain the concatenation x_2x_3 in the only synthesized argument in the left-hand side. Figure 18 shows the derivation tree that is induced by the sDCP rules on the given recursive partitioning (again abbreviating each Dutch word by its first letter). ■

We conclude:

Theorem 5
For each dependency structure h and recursive partitioning π of str(h), we can construct a LCFRS/sDCP hybrid grammar G such that G generates h and parses str(h) according to π.

6.7 Induction of LCFRS/sCFTG Hybrid Grammars from Dependency Structures

For dependency structures, we can define a notion of chunkiness similar to that in Section 6.5, relying on the definitions in Section 6.4. We say a recursive partitioning π with respect to dependency structure $h = (s, \leq_s)$ is **chunky** if for every node label J of π the length of $\top_{\max}(J)$ is 1. We have seen in Section 6.4 that the length of $\top_{\max}(J)$ determines the fanout of the corresponding nonterminal in the second component of

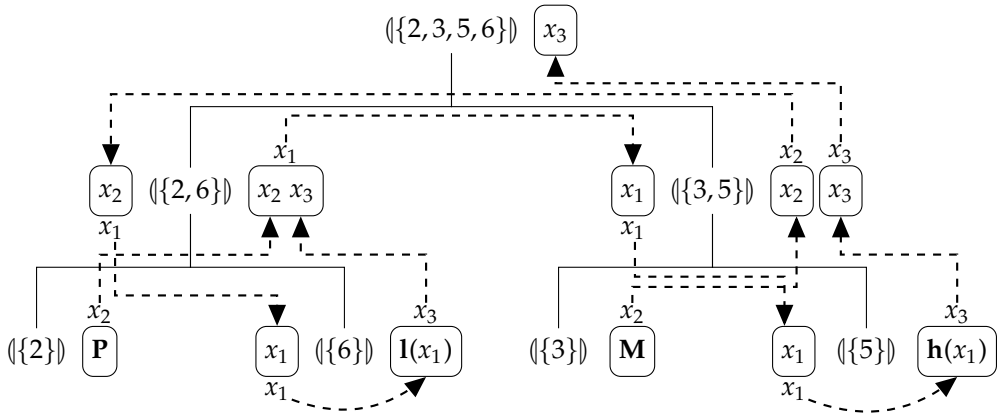

Figure 18
Derivation tree for the recursive partitioning of Figure 17, induced by the sDCP rules.

the constructed hybrid rule. We observed before that a sDCP grammar in which all nonterminals have s-rank 1 is equivalent to a sCFTG. We may conclude:

Theorem 6
For each dependency structure h and recursive partitioning π of str(h) that is chunky with respect to h, we can construct a LCFRS/sCFTG hybrid grammar G such that G generates h and parses str(h) according to π.

6.8 Induction on a Corpus

In the previous sections, we have induced a LCFRS/sDCP hybrid grammar G from a single phrase structure or dependency structure h. Given a training corpus c of phrase structures or dependency structures, we now want to induce a single hybrid grammar G that generalizes over the corpus. To this end, we apply one of the induction techniques to each hybrid tree h in c. The resulting grammars are condensed into a *single* hybrid grammar G by relabeling the existing nonterminals of the form $(\!|J|\!)$. This relabeling should be according to a naming scheme that is consistent to ensure that hybrid rules constructed from one hybrid tree for adjacent nodes of π can still link together to form a derivation. Beyond that, the naming scheme shall also allow for interaction of rules that were constructed from different hybrid trees, such that $L(G)$ contains meaningful hybrid trees that were not in c.

Two such naming schemes were considered in Nederhof and Vogler (2014) for a corpus of phrase structures. By **strict labeling**, a nonterminal name is chosen to consist of the terminal labels at the roots of the relevant subtrees of s. In Example 20, therefore, we would replace $(\!|\{1,2\}|\!)$ by $\langle$**hat**, **ADV**$\rangle$. (In a more realistic grammar, we would likely have a part of speech instead of **hat**.) This tends to lead to many nonterminal labels for different combinations of terminals. Therefore, an alternative was considered, called **child labeling**. This means that for two or more consecutive siblings in s, we collapse their sequence of terminals into a single tag of the form children-of(X), where X is the terminal label of the parent. This creates much fewer nonterminal names, but confuses different sequences of terminals that may occur below the same terminal. For more details, we refer the reader to Nederhof and Vogler (2014).

For grammar induction with a corpus of dependency structures, there is the additional complication of inherited arguments. If a LCFRS/sDCP hybrid grammar is induced from a single dependency structure, using nonterminals of the form $(\!|J|\!)$ as in Section 6.6, then cycles cannot occur in derivations of the sDCP. This is because, by definition, no cycles occur in the given dependency structure. However, if we replace nonterminals of the form $(\!|J|\!)$ by any other choice of symbols, and combine rules induced by different hybrid trees from a corpus, then cycles may arise if the relationship between inherited and synthesized arguments is confused.

One solution is to encode the dependencies between inherited and synthesized attributes into the nonterminal names—that is, for each synthesized attribute the list of inherited attributes from which it receives subtrees is specified (Angelov et al., 2014). Conceptually this is close to the g functions in the proof of Theorem 2 in Appendix C. One way to formalize this encoding is as follows.

As explained in Section 6.6, if we have a node label J in π, and $\bot_{\max}(J) = \langle O_1, \ldots, O_k \rangle$ and $\top_{\max}(J) = \langle I_1, \ldots, I_{k'} \rangle$, then this leads to creation of a nonterminal with k inherited arguments and k' synthesized arguments, so in total $k + k'$ arguments. We can construct a s-term σ in $T^*_{[k+k']}$, in which every number in $[k + k']$ occurs exactly once. In σ, argument numbers are located relative to one another as the corresponding positions in $\bot_{\max}(J) \cdot$

$\top_{\max}(J)$ are located in the hybrid tree. More precisely, if the i-th element and the j-th element of $\bot_{\max}(J) \cdot \top_{\max}(J)$ represent positions that share a parent, and those positions of the i-th precede those of the j-th, then number i precedes j in the root of the same sub-s-term of σ. Similarly, if the positions of the j-th element are descendants of at least one position in the i-th element, then j occurs as a descendant of i in σ.

Example 22
In combination with strict labeling, the nonterminal $\langle\!\{\{3,5\}\}\!\rangle$ in Example 21 could be replaced by $\langle$**Piet lezen**, **Marie**, **helpen**, σ$\rangle$, where σ is the s-term 3(1(2)). The "3" (third argument of the nonterminal) stands for the node in the hybrid tree labeled with **helpen**. Descendants of this node are the two nodes labeled with **Piet** and **lezen**, which belong to the first argument. The "2" (second argument) stands for the node labeled **Marie**, which is a descendant of **lezen** and therefore occurs below the "1." ■

7. Experiments

In this section, we present the first experimental results on induction of LCFRS/sDCP hybrid grammars from dependency structures and their application to parsing. We further present experiments on constituent parsing similar to those of Nederhof and Vogler (2014), but now with a larger portion of the TIGER corpus (Brants et al. 2004).

The purpose of the experiments is threefold: (i) A proof-of-concept for the induction techniques developed in Section 6 is provided. (ii) The influence of the strategy of recursive partitioning is evaluated empirically, as is the influence of the nonterminal naming scheme. We are particularly interested in how the strategy of recursive partitioning affects the size, parse time, accuracy, and robustness of the induced hybrid grammars. (iii) The performance of our architecture for syntactic parsing based on LCFRS/sDCP hybrid grammar is compared with two existing parsing architectures.

For all experiments, a corpus is split into a training set and a test set. A LCFRS/sDCP hybrid grammar is induced from the training set. Probabilities of rules are determined by relative frequency estimation. The induced grammar is then applied on each sentence of the test set (see Section 5.3), and the parse obtained from the most probable derivation is compared with the gold standard, resulting in a score for each sentence. The average of these scores for all test sentences is computed, weighted by sentence length.

All algorithms are implemented in Python and experiments are run on a server with two 2.6-GHz Intel Xeon E5-2630 v2 CPUs and 64 GB of RAM. Each experiment uses a single thread; the measured running time might be slightly distorted because of the usual load jitter. For probabilistic LCFRS parsing we use two off-the-shelf systems: If the induced grammar's first component is equivalent to a FA, then we use the OpenFST (Allauzen et al. 2007) framework with the Python bindings of Gorman (2016). Otherwise, we utilize the LCFRS parser of Angelov and Ljunglöf (2014), which is part of the runtime system of the Grammatical Framework (Ranta 2011).

7.1 Dependency Parsing

In our experiments on dependency parsing we use a corpus based on TIGER as provided in the 2006 CoNLL shared task (Buchholz and Marsi 2006). The task specifies splits of TIGER into a training set (39,216 sentences) and a test set (357 sentences). Each sentence in the corpus consists of a sequence of tokens. A token has up to 10 fields, including the sentence position, form, lemma, **part-of-speech** (POS) tag, sentence position of the head, and **dependency relation** (DEPREL) to the head. In TIGER, 52 POS tags and 46

DEPRELs are used. We adopt the three evaluation metrics from the shared task, namely, the percentages of tokens for which a parser correctly predicts the head (the **unlabeled attachment score** or UAS), the DEPREL (the **label accuracy** or LA), or both head and DEPREL (the **labeled attachment score** or LAS). Punctuation is removed from both training and test sets, and is thereby ignored for the purposes of these metrics. For testing we restrict ourselves to the 281 sentences of up to 20 (non-punctuation) tokens.

In our experiments, we use POS tags rather than lemmas or word forms. In order to accommodate for dependency relations, node labels are of the form (a, b), where a is the POS tag and b is the DEPREL to the head. Because input strings consist only of POS tags, each terminal symbol in the LCFRS-part of a rule is projected to its first component. Although our formal definition of hybrid rules only allows linking of pairs of terminal symbols that are identical, the definition can be suitably generalized, without changing the theory in any significant way. An example of a generalized hybrid rule, where a POS tag **NN** is linked to a node label (**NN**, **dobj**), is:

$$[(\!|\{5\}|\!)(\mathbf{NN}^{\boxed{1}}) \to \langle\rangle,\ \ (\!|\{5\}|\!)(x_1, (\mathbf{NN}, \mathbf{dobj})^{\boxed{1}}(x_1)) \to \langle\rangle]$$

Our experiments with grammar induction are controlled by three parameters, each ranging over a number of values:

- The *naming scheme* for nonterminals can be (i) strict labeling or (ii) child labeling, as outlined in Section 6.8.
- In both naming schemes, sequences of terminal labels from the hybrid tree are composed into nonterminal labels. For each terminal, which is a CoNLL token, we include only particular fields, namely, (i) POS and DEPREL, (ii) POS, or (iii) DEPREL. We call this parameter **argument label**.
- The considered methods to obtain *recursive partitionings* include (i) direct extraction (see Algorithm 3), (ii) transformation to fanout k (see Algorithm 4), (iii) right-branching, and (iv) left-branching.

Each choice of values for the parameters determines an **experimental scenario**. In particular, direct extraction in combination with strict labeling achieves traditional LCFRS parsing.

We compare our induction framework with rparse (Maier and Kallmeyer 2010), which induces and trains unlexicalized, binarized, and Markovized LCFRS. As parser for these LCFRS we choose the runtime system of the Grammatical Framework, because it is faster than rparse's built-in parser (for a comparison cf. Angelov and Ljunglöf 2014).

Additionally, we use MaltParser (Nivre, Hall, and Nilsson 2006) to compare our approach with a well-established transition-based parsing architecture, which is not state-of-the-art but allows us to disable features of lemmas and word forms, to match our own implementation, which does not take lemmas or word forms as part of the input. The stacklazy strategy (Nivre, Kuhlmann, and Hall 2009) with the LibLinear classifier is used.

Experimental Results. Statistics on the induced hybrid grammars and parsing results are listed in Table 2. For the purpose of measuring UAS, LAS, and LA, we take a default dependency structure in the case of a parse failure; in this default structure, the head of the i-th word is the $i - 1$-th word, and the DEPREL fields are left empty. Figure 19

Table 2
Experiments with dependency parsing (TIGER in CoNLL 2006 shared task): method of recursive partitioning for extraction, argument label, number of nonterminals and rules, maximum and average fanout, number of parse failures, unlabeled attachment score, labeled attachment score, label accuracy, and parse time in seconds.

extraction	arg. lab.	nont.	rules	$f_{\max}$	f_{avg}	fail	UAS	LAS	LA	time
child labeling										
direct	POS+DEPREL	4,043	61,923	4	1.06	68	68.0	59.5	63.2	97
$k = 1$	POS+DEPREL	17,333	60,399	1	1.00	9	**85.8**	**79.7**	85.5	94
$k = 2$	POS+DEPREL	10,777	49,448	2	1.18	11	85.7	**79.7**	85.4	103
$k = 3$	POS+DEPREL	10,381	48,844	3	1.19	11	85.6	**79.7**	85.3	108
r-branch	POS+DEPREL	92,624	191,341	1	1.00	60	68.9	60.3	65.0	27
l-branch	POS+DEPREL	96,980	196,125	1	1.00	56	70.3	61.9	66.7	28
direct	POS	799	43,439	4	1.10	23	78.2	59.7	67.2	113
$k = 1$	POS	6,060	28,880	1	1.00	4	83.2	65.3	73.2	70
$k = 2$	POS	2,396	20,592	2	1.38	4	83.8	65.4	73.1	86
$k = 3$	POS	2,121	20,100	3	1.44	4	83.4	65.2	73.1	88
r-branch	POS	47,661	123,367	1	1.00	23	77.9	59.8	68.1	53
l-branch	POS	49,203	125,406	1	1.00	26	78.5	60.2	67.5	51
direct	DEPREL	527	33,844	4	1.10	1	79.5	72.4	82.7	251
$k = 1$	DEPREL	4,344	21,613	1	1.00	1	78.0	70.6	81.4	172
$k = 2$	DEPREL	1,739	15,184	2	1.35	1	78.7	70.9	81.6	263
r-branch	DEPREL	40,239	99,113	1	1.00	1	77.2	69.1	80.7	94
l-branch	DEPREL	37,535	92,390	1	1.00	1	78.1	69.8	80.9	85
strict labeling										
direct	POS+DEPREL	48,404	106,284	4	1.01	68	68.0	59.5	63.2	122
$k = 1$	POS+DEPREL	103,425	162,124	1	1.00	57	72.2	64.2	68.3	185
$k = 2$	POS+DEPREL	92,395	150,013	2	1.08	55	72.0	64.4	68.6	266
$k = 3$	POS+DEPREL	91,412	149,106	3	1.08	55	72.0	64.4	68.6	240
r-branch	POS+DEPREL	251,536	338,294	1	1.00	141	44.6	31.4	32.7	54
l-branch	POS+DEPREL	264,190	349,299	1	1.00	137	45.3	32.6	34.2	53
direct	POS	29,165	80,695	4	1.00	23	77.6	59.6	67.5	120
$k = 1$	POS	62,769	115,363	1	1.00	18	78.7	60.9	69.1	201
$k = 2$	POS	54,082	104,390	2	1.09	18	79.5	61.3	69.4	237
$k = 3$	POS	53,186	103,503	3	1.11	18	79.6	61.4	69.5	231
r-branch	POS	181,432	277,201	1	1.00	98	55.1	36.4	40.8	88
l-branch	POS	190,890	286,273	1	1.00	108	52.2	33.9	37.7	87
direct	DEPREL	17,047	53,342	4	1.00	3	82.4	76.5	84.6	178
$k = 1$	DEPREL	37,333	71,423	1	1.00	1	83.2	78.0	**85.8**	188
$k = 2$	DEPREL	31,956	63,487	2	1.08	2	83.0	77.5	85.5	231
r-branch	DEPREL	126,841	197,261	1	1.00	2	80.8	74.4	83.1	101
l-branch	DEPREL	124,722	192,922	1	1.00	2	81.2	74.8	83.5	96
rparse ($v = 1, h = 5$)		46,799	72,962	5	1.07	2	85.3	79.0	86.4	228
MaltParser, unlexicalized, stacklazy						0	88.2	83.7	88.7	2

shows the distributions of the fanout of nonterminals in the LCFRS and the numbers of arguments in the sDCP, depending on the recursive partitioning strategy if we fix child labeling with POS+DEPREL as argument labels. In the following we discuss trends that can be observed in these data if we change the value of one parameter while keeping others fixed.

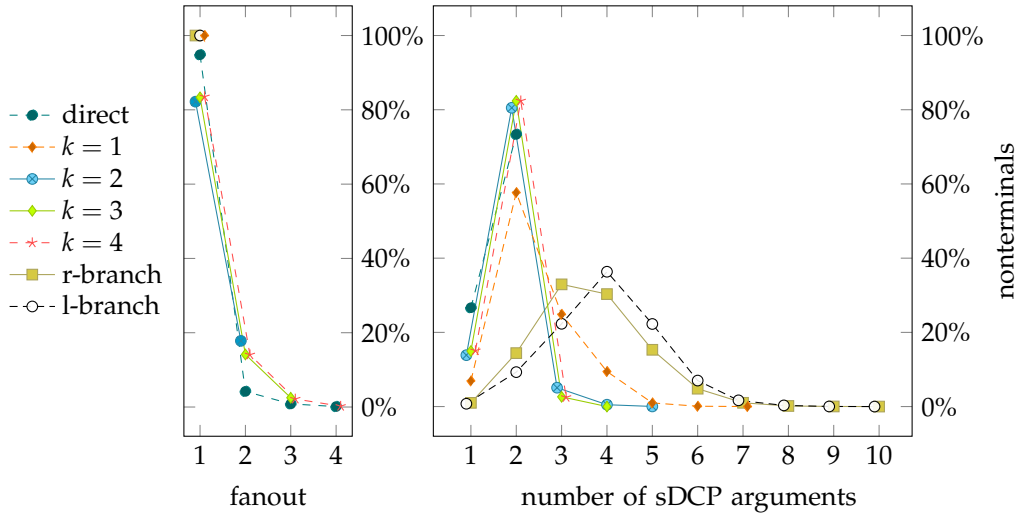

Figure 19
The percentage of nonterminals with a particular fanout and number of sDCP arguments depending on the recursive partitioning strategy. All grammars are induced from TIGER using the child labeling strategy with POS and DEPREL as argument labels.

Naming Scheme. As expected, child labeling leads to significantly fewer distinct nonterminals than strict labeling, by a factor of between 2.5 and 37, depending on the other parameter values. This makes the tree language less refined, and one may expect the scores therefore to be generally lower. However, in many cases where strict labeling suffers from a higher proportion of parse failures (so that the parser must fall back on the default structure), it is child labeling that has the higher scores.

Argument Labels. Including only POS tags in nonterminal labels generally leads to high UAS but low LA. By including DEPRELs but not POS tags, the LA and LAS are higher in all cases. For child labeling, this is at the expense of a lower UAS, in all cases except one. For strict labeling, UAS is higher in all cases. There are also fewer parse failures, which may be because the number of DEPRELs is smaller than the number of POS tags.

With the combination of POS tags and DEPRELs, the tree language can be most accurately described, and we see that this achieves some of the highest UAS and LAS in the case of child labeling. However, the scores can be low in the presence of many parse failures, due to the fall-back on the default structure. This holds in particular in the case of strict labeling.

Recursive Partitionings. Concerning the choice of the method to obtain recursive partitionings, the baseline is direct extraction, which produces an unrestricted LCFRS in the first component. For instance, for child labeling and POS and DEPREL as argument labels, the induced LCFRS has fanout 4 and about 88% of the rules have three or more nonterminals on the right-hand side. In total there are 4,043 nonterminals, the majority of which have fanout 1. Reduction of the fanout to $1 \leq k \leq 3$ leads to a binarized grammar with smaller fanout as desired but the number of nonterminals is quadrupled. By transforming a recursive partitioning with parameter $k \geq 2$, the average fanout f_{avg} of nonterminals may in fact increase, which is somewhat counter-intuitive. Whereas the distribution of nonterminals with fanout 1, 2, 3 is 94.8%, 4.2%, 0.9% in the case of direct extraction,

this changes to distribution 83.4%, 14.2%, 2.5% for the transformation with $k = 2$. We suspect the increase is due to the high fanout of newly introduced nodes (see line 8 of Algorithm 4).

We can further see in Figure 19 that the number of arguments in the sDCP increases signficantly once the fanout is restricted to $k = 1$. The binarization involved in the process of reducing the fanout to some value of k seems to improve the ability of the grammar to generalize over the training data, as here both the number of parse failures drops and the scores increase. The exact choice of k has little impact on the scores, however.

The measured parse times in most cases increase if higher values of k are chosen. In some cases, however, the parse times are lower for direct extraction. This may be explained by the higher average fanout and differences in the sizes of the grammars.

The left-branching and right-branching recursive partitionings lead to many specialized nonterminal symbols (and rules). In comparison with the partitioning strategies discussed earlier, we observe that a nonterminal can have up to 10 sDCP arguments, the average lying between 3 and 4. The scores are often worse and there tend to be more parse failures. However, in the case of child labeling with DEPREL as argument label, the scores are very similar. Left-branching seems to lead to slightly higher scores than right-branching recursive partitioning, except for POS as argument label. With left-branching and right-branching recursive partitionings, parsing tends to be faster than with the other recursive partitionings. However, in many cases we do not observe the predicted asymptotic differences, which may be due to larger grammar sizes.

Overall, child labeling with POS+DEPREL and $k = 1$ and strict labeling with DEPREL and $k = 1$ turned out to be the best choices for maximizing UAS/LAS and LA, respectively. The former scenario slightly outperforms rparse with respect to UAS and LAS and parse time whereas rparse obtains better LA. Still, MaltParser outperforms these experimental scenarios with respect to both the obtained scores and the measured parse times. Our restriction to sentence length 20, which we mentioned earlier, was motivated by the excessive computational costs for instances with high (average) fanout.

The most suitable choices for naming scheme, argument labeling, and recursive partitioning may differ between languages and between annotation schemes. For instance, for the NEGRA (Skut et al. 1997) benchmark used by Maier and Kallmeyer (2010), we obtain the results in Table 3, which suggest the combination of child labeling, DEPREL, and $k = 2$ is the most suitable choice if LAS is the target score. It appears that scenarios with POS+DEPREL give lower scores than DEPREL, mainly because of the many parse failures, despite the fact that such argument labeling can be expected to lead to more refined tree languages. For this reason, we also consider a cascade of three scenarios with child labeling, $k = 1$, and argument labels set to POS+DEPREL, POS, or DEPREL, respectively, where in the case of parse failure we fall back to the next scenario. This cascade achieves better scores than any individual experimental scenario and the additional parse time with respect to POS+DEPREL alone is small. Both the single best scenario and the cascade perform better than the baseline LCFRS (rparse simple) induced with the algorithm by Kuhlmann and Satta (2009) and the LCFRS that is obtained from the baseline by binarization, vertical Markovization $v = 1$, and horizontal Markovization $h = 3$. Again, hybrid grammars fall short of MaltParser. We do not include results for NEGRA with the strict labeling strategy, because the numbers of parse failures are too high and, consequently, accuracies are too low to be of interest.

Note that the hybrid grammar obtained with child labeling, DEPREL, and direct extraction should, in principle, be similar to the baseline LCFRS. Indeed, both grammars have the same fanout and similar numbers of rules. The differences in size can be explained by the separation of terminal generating and structural rules during hybrid

Table 3
Experiments with dependency parsing (NEGRA). From the sentences with length ≤ 25, the first 14,858 make up the training corpus and the remaining 1,651 the test corpus. Additionally to the columns in Table 2, we present unlabeled and labeled attachment scores that include punctuation (UASp and LASp, respectively).

extraction	arg. lab.	nont.	rules	$f_{\max}$	f_{avg}	fail	UASp	LASp	UAS	LAS	LA	time
child labeling												
direct	P+D	4,739	27,042	7	1.10	693	51.7	40.7	52.2	40.8	42.6	253
$k = 1$	P+D	13,178	35,071	1	1.00	202	75.9	68.7	77.0	69.1	73.3	288
$k = 2$	P+D	11,156	32,231	2	1.17	195	76.5	69.6	77.7	70.1	74.2	355
r-branch	P+D	42,577	79,648	1	1.00	775	45.5	33.1	45.7	32.8	34.7	49
l-branch	P+D	40,100	75,321	1	1.00	768	45.8	33.4	46.0	33.2	35.0	45
direct	POS	675	19,276	7	1.24	303	68.7	51.5	69.3	50.0	55.4	300
$k = 1$	POS	3,464	15,826	1	1.00	30	**81.7**	65.5	**82.5**	63.5	70.6	244
$k = 2$	POS	2,099	13,347	2	1.40	35	81.6	65.2	82.4	63.3	70.5	410
r-branch	POS	19,804	51,733	1	1.00	372	62.7	46.4	62.7	44.7	50.6	222
l-branch	POS	17,240	45,883	1	1.00	342	63.7	47.4	63.9	45.6	51.4	197
direct	DEP	2,505	19,511	7	1.13	3	78.5	**72.2**	78.9	71.6	78.6	484
$k = 1$	DEP	8,059	22,613	1	1.00	1	78.5	71.7	79.5	71.7	79.0	608
$k = 2$	DEP	6,651	20,314	2	1.20	1	78.7	72.1	79.8	**72.0**	**79.2**	971
$k = 3$	DEP	6,438	19,962	3	1.25	1	78.6	72.0	79.5	71.9	79.1	1,013
r-branch	DEP	27,653	54,360	1	1.00	2	76.0	68.4	76.3	67.5	76.1	216
l-branch	DEP	25,699	50,418	1	1.00	1	75.8	68.4	76.2	67.6	76.1	198
cascade: child labeling, $k = 1$, P+D/POS/DEP						1	**83.2**	**76.2**	**84.3**	**76.1**	**81.6**	325
LCFRS (Maier and Kallmeyer 2010)						-	79.0	71.8	-	-	-	-
rparse simple		920	18,587	7	1.37	56	77.1	70.6	77.3	70.0	76.2	350
rparse ($v = 1, h = 3$)		40,141	61,450	7	1.10	13	78.4	72.2	78.5	71.4	79.0	778
MaltParser, unlexicalized, stacklazy						0	85.0	80.2	85.6	80.0	85.0	24

grammar induction, which is not present in the induction algorithm by Kuhlmann and Satta (2009). This separation also leads to different generalization over the training data, as the higher accuracy and the lower numbers of parse failures for the hybrid grammar indicate.

One possible refinement of this result is to choose different argument labels for inherited and synthesized arguments. One may also use form or lemma next to, or instead of, POS tags and DEPRELs, possibly in combination with smoothing techniques to handle unknown words. Another subject for future investigation is the use of splitting and merging (Petrov et al. 2006) to determine parts of nonterminal names. New recursive partitioning strategies may be developed, and one could consider blending grammars that were induced using different recursive partitioning strategies.

7.2 Constituent Parsing

The experiments for constituent parsing are carried out as in Nederhof and Vogler (2014) but on a larger portion of the TIGER corpus: We now use the first 40,000 sentences for training (omitting 10 where a single tree does not span the entire sentence), and from the remaining 10,474 sentences we remove the ones with length greater than 20, leaving 7,597 sentences for testing. The results are displayed in Table 4 and allow for similar conclusions as in Nederhof and Vogler (2014). Note that for direct extraction both labeling

Table 4
Experiments for constituent parsing (TIGER). Number of nonterminals, rules, and parse failures, recall, precision, F-measure, average number of gaps per constituent, and parse time in seconds.

	nont.	rules	fail	R	P	F1	# gaps	time
strict labeling								
direct	104	31,656	10	77.5	77.7	76.9	0.0139	2,098
$k = 1$	26,936	62,221	11	77.1	77.1	76.4	0.0136	2,514
$k = 2$	19,387	51,414	9	77.5	77.8	76.9	0.0136	2,892
$k = 3$	18,685	50,678	9	77.5	77.8	76.9	0.0135	2,886
r-branch	164,842	248,063	658	61.2	58.2	59.1	0.0135	1,341
l-branch	284,816	348,536	3,662	37.7	34.8	35.7	0.0131	2,143
child labeling								
$k = 1$	2,117	13,877	1	75.3	74.9	74.5	0.0140	1,196
$k = 2$	473	8,078	1	75.6	75.4	74.9	0.0144	1,576
$k = 3$	176	7,352	1	75.7	75.4	74.9	0.0144	1,667
r-branch	27,222	83,035	36	75.0	74.4	74.1	0.0148	502
l-branch	87,171	162,645	137	74.5	73.9	73.6	0.0146	702

strategies yield the same grammar; thus, only one entry is shown. Unsurprisingly, the larger training set leads to smaller proportions of parse failures and to improvements of F-measure. Another consequence is that the more fine-grained strict labeling now outperforms child labeling, except in the case of right-branching and left-branching, where the numbers of parse failures push the F-measure down.

Note that the average numbers of gaps per constituent are very similar for the different recursive partitionings. One may observe once more that the parse times of right-branching and left-branching recursive partitionings are higher than one might expect from the asymptotic time complexities. This is again due to the considerable sizes of the grammars.

8. Related Work

Two kinds of discriminative models of non-projective dependency parsing have been intensively studied. One is based on an algorithm for finding the maximum spanning tree (MST) of a weighted directed graph, where the vertices are the words of a sentence, and each edge represents a potential dependency relation (McDonald et al. 2005). In principle, any non-projective structure can be obtained, depending on the weights of the edges. A disadvantage of MST dependency parsing is that it is difficult to put any constraints on the desirable structures beyond the local constraints encoded in the edge weights (McDonald and Pereira 2006).

The second kind of model involves stack-based transition systems, with an added transition that swaps stack elements. In particular, Nivre (2009) introduced a deterministic system that uses a classifier to determine the next transition to be applied. The worst-case time complexity is quadratic, and the expected complexity is linear. The classifier relies on features that look at neighboring words in the sentence, as well as at vertical and horizontal context in the syntactic tree. Advances in learning the relevant features are due to Chen and Manning (2014).

In this article we have assumed a generative model of hybrid grammars, which differs from deterministic, stack-based models in at least two ways, one of which is superficial whereas the other is more fundamental. The superficial difference is the

presence of nonterminals in hybrid grammars. These, however, fulfill a role that is comparable to that of sets of features used by classifiers. We conjecture that machine learning techniques could even be introduced to create more refined nonterminals for hybrid grammars. The more fundamental difference lies in the determinism of the discussed stack-based models, which is difficult to realize for hybrid grammars, except perhaps in the case of left-branching and right-branching recursive partitionings. Without determinism, the time complexity grows with the fanout that we allow for the left components of hybrid grammars. What we get in return for the higher running time are more powerful models for parsing the input.

As in the case of MST dependency parsing, the discussed stack-based transition systems can in principle produce any non-projective structure. The non-projectivity allowed by hybrid grammars is determined by the hybrid rules, which in turn are determined by non-projectivity that occurs in the training data. This means that there is no restriction per se on non-projectivity in structures produced by a parser for test data, provided the training data contains an adequate amount of non-projectivity. This even holds if we restrict the fanout of the first component, although we may then need more training data to obtain the same coverage and accuracy.

Many established algorithms for constituent parsing rely on generative models and grammar induction (Collins 1997; Charniak 2000; Klein and Manning 2003; Petrov et al. 2006). Most of these are unable to produce discontinuous structures. A notable exception is Maier and Søgaard (2008), where the induced grammar is a LCFRS. Such a grammar can be seen as a special case of a LCFRS/sDCP hybrid grammar, with the restriction that each nonterminal has a single synthesized argument. This restriction limits the power of hybrid grammars. In particular, discontinuous structures can now only be produced if the fanout is strictly greater than 1, which also implies the time complexity is more than cubic.

Generative models proposed for dependency parsing have often been limited to projective structures, based on either context-free grammars (Eisner 1996; Klein and Manning 2004) or tree substitution grammars (Blunsom and Cohn 2010). Exceptions are recent models based on LCFRS (Maier and Kallmeyer 2010; Kuhlmann 2013). As in the case of constituent parsing, these can be seen as restricted LCFRS/sDCP hybrid grammars.

A related approach is from Satta and Kuhlmann (2013). Although it does not use an explicit grammar, there is a clear link to mildly context-sensitive grammar formalisms, in particular lexicalized TAG, following from the work of Bodirsky, Kuhlmann, and Möhl (2005).

The flexibility of generative models has been demonstrated by a large body of literature. Applications and extensions of generative models include syntax-based machine translation (Charniak, Knight, and Yamada 2003), discriminative reranking (Collins 2000), and involvement of discriminative training criteria (Henderson 2004). It is very likely that these apply to hybrid grammars as well. Further discussion is outside the scope of this article.

Appendix A. Single-Synthesized sDCPs Have the Same s-term Generating Power as sCFTGs

The proof of Theorem 1 is the following.

Proof. Let G be a single-synthesized sDCP. We construct a sCFTG G' such that $[G] = [G']$. For each nonterminal A from G, there will be one nonterminal A' in G', with $\mathrm{rk}(A') = \text{i-rk}(A)$.

Let $A_0(x^{(0)}_{1,k_0}, s^{(0)}) \to \langle A_1(s^{(1)}_{1,k_1}, x^{(1)}), \ldots, A_n(s^{(n)}_{1,k_n}, x^{(n)})\rangle$ be a rule of G, with $k_m = \text{i-rk}(A_m)$ for $m \in [n]_0$. Assume without loss of generality that $x^{(0)}_{1,k_0} = x_{1,k_0}$. Then G' will contain the rule $A'_0(x_{1,k_0}) \to \text{rhs}(s^{(0)})$, where rhs is defined by:

$$\begin{aligned}
\text{rhs}(\langle t_{1,k}\rangle) &= \text{rhs}(t_1) \cdot \ldots \cdot \text{rhs}(t_k)\\
\text{rhs}(\delta(s_{1,k})) &= \langle \delta(\text{rhs}(s_1), \ldots, \text{rhs}(s_k))\rangle\\
\text{rhs}(x) &= \langle x\rangle, \text{ if } x \text{ is } x^{(0)}_i \ (i \in [k_0])\\
\text{rhs}(x) &= A'_m(\text{rhs}(s^{(m)}_1), \ldots, \text{rhs}(s^{(m)}_{k_m})), \text{ if } x \text{ is } x^{(m)} \ (m \in [n])
\end{aligned}$$

It is not difficult to prove that:

$$\langle A(s_{1,k}, s)\rangle \Rightarrow^*_G \langle\rangle \text{ if and only if } \langle A'(s_{1,k})\rangle \Rightarrow^*_{G'} s$$

for every nonterminal A of G and s-terms $s, s_1, \ldots, s_k$. Hence $[G] = [G']$.

There is an inverse transformation from a sCFTG to a sDCP with s-rk$(A) = 1$ for each A. This is as straightforward as the transformation above. In fact, one can think of sCFTGs and sDCPs with s-rank restricted to 1 as syntactic variants of one another. ■

A more extensive treatment of a very closely related result is from Mönnich (2010), who refers to a special form of attributed tree transducers in place of sDCPs. This result for tree languages was inspired by an earlier result by Duske et al. (1977) for string languages, involving (non-simple) macro grammars (with IO derivation) and simple-L-attributed grammars. For arbitrary s-rank, a related result is that attributed tree transductions are equivalent to attribute-like macro tree transducers, as shown by Fülöp and Vogler (1999).

Appendix B. The Class of s-term Languages Induced by sDCP Is Strictly Larger than that Induced by sCFTG

By Theorem 1, sCFTG and single-synthesized sDCP have the same s-term generating power. Thus it suffices to show that the full class of sDCP is strictly more powerful than single-synthesized sDCP. For this we represent the only-synthesized attribute grammar of Engelfriet and Filè (1981, page 298) as sDCP G as follows:

$$\begin{aligned}
S(\mathbf{A}(x_1\, x_2)) &\to A(x_1, x_2)\\
A(\mathbf{B}(x_1\, x_3), \mathbf{B}'(x_2\, x_4)) &\to A(x_1, x_2)\, A(x_3, x_4)\\
A(\mathbf{B}(\varepsilon), \mathbf{B}'(\varepsilon)) &\to \varepsilon
\end{aligned}$$

where s-rk$(S) = 1$, s-rk$(A) = 2$, and i-rk$(A) = 0$. It should be clear that the induced s-term language $[G]$ contains trees of the form $\mathbf{A}(s, s')$ where s and s' are s-terms over $\{\mathbf{B}\}$ and $\{\mathbf{B}'\}$, respectively, and $\text{pos}(s) = \text{pos}(s')$ and s' is obtained from s by replacing each $\mathbf{B}$ by $\mathbf{B}'$.

In Section 5 of Engelfriet and Filè (1981) it was proved that $[G]$, viewed as language of binary trees, cannot be induced by 1S-AG, that is, attribute grammars with one synthesized attribute (and any number of inherited attributes). Although single-synthesized sDCP can generate s-terms (and not only trees as 1S-AG can), it is rather

obvious to see that this extra power does not help to induce $[G]$. Thus we conclude that there is no single-synthesized sDCP that induces $[G]$.

Appendix C. sDCP Have the Same String Generating Power as LCFRS

The proof of Theorem 2 is the following.

Proof. We show that for each sDCP G_1 there is a LCFRS G_2 such that $[G_2] = \mathrm{pre}([G_1])$, where pre was extended from s-terms to sets of s-terms in the obvious way. Our construction first produces sDCP G_1' from G_1 by replacing every s-term s in every rule of G_1 by $\mathrm{pre}(s)$. It is not difficult to see that $[G_1'] = \mathrm{pre}([G_1])$.

Next, for every nonterminal A in G_1', with $\text{i-rk}(A) = k'$ and $\text{s-rk}(A) = k$, we introduce nonterminals of the form $A^{(g)}$, where g is a mapping from $[k]$ to sequences of numbers in $[k']$, such that each $j \in [k']$ occurs precisely once in $g(1) \cdot \ldots \cdot g(k)$. The intuition is that if $g(i) = \langle j_1, \ldots, j_{p_i} \rangle$, then a value appearing as the j_q-th inherited argument of A reappears as part of the i-th synthesized argument, and it is the q-th inherited argument to do so. (This concept is similar to the argument selector in Courcelle and Franchi-Zannettacci [1982, page 175].) For each A, only those functions g are considered that are consistent with at least one subderivation of G_1'.

We will show that whereas replacing a nonterminal A_m by $A_m^{(g_m)}$, as m-th member in the right-hand side of a rule of the form of Equation (3), a variable $x_i^{(m)}$ appearing as the i-th synthesized argument is split up into several new variables $x_i^{(m,0)}, \ldots, x_i^{(m,p_i^{(m)})}$. These variables are drawn from X so as not to clash with variables used before. The $p_i^{(m)}$ terms of the inherited arguments whose indices are listed by $g_m(i)$ will be shifted to the left-hand side of a new rule to be constructed, interspersed with the $p_i^{(m)} + 1$ new variables. This shifting of terms may happen along dependencies between inherited and synthesized arguments of other members in the right-hand side. (This shifting is again very similar to the one for obtaining rules for an IO-macro grammar from a simple L-attributed grammar (Duske et al. 1977, Def.6.1)).

More precisely, we construct a new grammar G_1'' from G_1' by iteratively adding more elements to its set of nonterminals and to its set of rules, until no more new nonterminals and rules can be found. In each iteration, we consider a rule from G_1' of the form $A_0(x_{1,k_0'}^{(0)}, s_{1,k_0}^{(0)}) \to \langle A_1(s_{1,k_1'}^{(1)}, x_{1,k_1}^{(1)}), \ldots, A_n(s_{1,k_n'}^{(n)}, x_{1,k_n}^{(n)}) \rangle$, and a choice of functions $g_1, \ldots, g_n$ such that the nonterminals $A_m^{(g_m)}$ ($m \in [n]$) were found before. We define a function "exp" that expands terms and s-terms as follows:

$$\begin{aligned}
\exp(\langle t_1, \ldots, t_k \rangle) &= \exp(t_1) \cdot \ldots \cdot \exp(t_k) \\
\exp(\delta(\langle\rangle)) &= \langle \delta(\langle\rangle) \rangle \\
\exp(x) &= \langle x \rangle, \text{ if } x \text{ is } x_i^{(0)} \ (i \in [k_0']) \\
\exp(x) &= \langle x_i^{(m,0)} \rangle \cdot \exp(s_{j_1}^{(m)}) \cdot \langle x_i^{(m,1)} \rangle \cdot \ldots \cdot \exp(s_{j_{p_i^{(m)}}}^{(m)}) \cdot \langle x_i^{(m,p_i^{(m)})} \rangle \\
&\qquad \text{if } x \text{ is } x_i^{(m)} \ (m \in [n], i \in [k_m]) \text{ and } g_m(i) = \langle j_1, \ldots, j_{p_i^{(m)}} \rangle
\end{aligned}$$

Note that, despite the recursion, "exp" is well-defined due to our assumption that sDCPs are free of cycles.

For $i \in [k_0]$, define $g_0(i) = \langle j_1, \cdots, j_{p_i} \rangle$, where $\exp(s_i^{(0)})$ is of the form:

$$r_0^{(i)} \cdot \langle x_{j_1}^{(0)} \rangle \cdot r_1^{(i)} \cdot \ldots \cdot \langle x_{j_{p_i}}^{(0)} \rangle \cdot r_{p_i}^{(i)}$$

and no variables from among $x_{1,k_0'}^{(0)}$ occur in any $r_j^{(i)}$ ($j \in [p_i]_0$).

The new grammar G_1'' will now contain the rule:

$$A_0^{(g_0)}(r_{0,p_1}^{(1)}, \ldots, r_{0,p_{k_0}}^{(k_0)}) \to \langle A_1^{(g_1)}(y_{1,k_1}^{(1)}), \ldots, A_n^{(g_n)}(y_{1,k_n}^{(n)}) \rangle$$

where each $y_i^{(m)}$ is $x_i^{(m,0)}$, …, $x_i^{(m,p_i^{(m)})}$, for $m \in [n]$ and $i \in [k_m]$. This also defines the nonterminal $A_0^{(g_0)}$ to be added to G_1'' if it does not already exist.

In order to clarify the semantic relationship between derivations of G_1' and G_1'' we allow the sentential forms in derivations of a sDCP to contain variables; they can be viewed as extra nullary symbols. Moreover, each derivation of the form $\langle A(x_{1,k_0'}, s_{1,k_0}) \rangle \Rightarrow_{G_1'}^* \langle\rangle$ induces in an obvious way an argument selector function g. Let A be a nonterminal with k_0' inherited arguments and k_0 synthesized arguments. Let s_{1,k_0} be s-terms in $T_\Sigma(X_{k_0'})$ such that each $x_i \in X_{k_0'}$ occurs exactly once in s_{1,k_0}. For each $j \in [k_0]$ we let:

$$s_j = r_0^{(j)} \cdot \langle x_{j_1} \rangle \cdot r_1^{(j)} \cdot \ldots \cdot \langle x_{j_{p_j}} \rangle \cdot r_{p_j}^{(j)}$$

such that none of $r_0^{(j)}, \ldots, r_{p_j}^{(j)}$ contain any variables. We define the argument selector mapping g by $g(j) = \langle j_1, \ldots, j_{p_j} \rangle$ for each $j \in [k_0]$.

Then the following two statements are equivalent:

1. $\langle A(x_{1,k_0'}, s_{1,k_0}) \rangle \Rightarrow_{G_1'}^* \langle\rangle$ and g is the argument selector function of this derivation.
2. $\langle A^{(g)}(r_{0,p_1}^{(1)}, \ldots, r_{0,p_{k_0}}^{(k_0)}) \rangle \Rightarrow_{G_1''}^* \langle\rangle$.

The grammar G_1'' constructed here has no inherited arguments and is almost the required LCFRS G_2. To precisely follow our definitions, what remains is to consistently rename the variables in each rule to obtain the set X_m, some m. Furthermore, the symbols from Σ now need to be explicitly assigned rank 0.

For the converse direction of the theorem consider LCFRS G. We construct sDCP G' by taking the same nonterminals as those of G, with the same ranks. Each argument is synthesized. To obtain the rules of G', we replace each term $a()$ by the term $a(\langle\rangle)$. ■

Example 23
If G_1 contains the rules:

$$C(x_1, x_2, \sigma(x_2\alpha(x_1))) \to \varepsilon$$
$$A(x_1, x_2, \alpha(x_1x_3)) \to B(\beta(x_2), x_4)\, C(\gamma, x_4, x_3)$$

then G_1' has the rules:

$$C(x_1, x_2, \sigma\ x_2\ \alpha\ x_1) \to \varepsilon$$
$$A(x_1, x_2, \alpha\ x_1\ x_3) \to B(\beta\ x_2, x_4)\ C(\gamma, x_4, x_3)$$

where i-rk$(A) = 2$, i-rk$(B) = 1$, and i-rk$(C) = 2$ for G_1 and G_1'. For the first rule we have $\exp(\sigma\ x_2\ \alpha\ x_1) = \sigma\ x_2\ \alpha\ x_1$ and we obtain the function $g(1) = \langle 2, 1\rangle$, because x_2 occurs before x_1. This leads to the new rule:

$$C^{(g)}(\sigma, \alpha, \varepsilon) \to \varepsilon$$

For the second rule in G_1', assume $g_2 = g$ with g the function we found for the first rule. Further assume we previously found nonterminal $B^{(g_1)}$ with $g_1(1) = \langle 1\rangle$, which is in fact the only allowable function for B. Then:

$$\begin{aligned}
&\exp(\alpha x_1 x_3)\\
&= \alpha x_1 \exp(x_3)\\
&= \alpha x_1 x_3^{(0)} \exp(x_4) x_3^{(1)} \exp(\gamma) x_3^{(2)}\\
&= \alpha x_1 x_3^{(0)} x_4^{(0)} \exp(\beta x_2) x_4^{(1)} x_3^{(1)} \gamma x_3^{(2)}\\
&= \alpha x_1 x_3^{(0)} x_4^{(0)} \beta x_2 x_4^{(1)} x_3^{(1)} \gamma x_3^{(2)}
\end{aligned}$$

Because x_1 occurs before x_2 we have $g_0(1) = \langle 1, 2\rangle$ and we derive the rule:

$$A^{(g_0)}(\alpha, x_3^{(0)} x_4^{(0)} \beta, x_4^{(1)} x_3^{(1)} \gamma x_3^{(2)}) \ \to\ B^{(g_1)}(x_4^{(0)}, x_4^{(1)})\ C^{(g_2)}(x_3^{(0)}, x_3^{(1)}, x_3^{(2)})$$ ■

Example 24
By the same construction, the sDCP from Example 8 is transformed into the following LCFRS (after renaming of variables, and simplifying nonterminal names):

$$S'(x_2 x_1 x_3) \to A'(x_1)\ B'(x_2, x_3) \qquad B'(\mathbf{c}\ \mathbf{B}\ x_1, x_2\ \mathbf{d}) \to B'(x_1, x_2)$$
$$A'(\mathbf{a}\ \mathbf{A}\ x_1\ \mathbf{b}) \to A'(x_1) \qquad B'(\varepsilon, \varepsilon) \to \varepsilon$$
$$A'(\varepsilon) \to \varepsilon$$ ■

Acknowledgments
We are grateful to the reviewers for their constructive criticism and encouragement. We also thank Markus Teichmann for helpful discussions. The third author was financially supported by the Deutsche Forschungsgemeinschaft by project DFG VO 1011/8-1.

References

Allauzen, Cyril, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. Openfst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata: 12th International Conference*, pages 11–23, Prague.

Angelov, Krasimir and Peter Ljunglöf. 2014. Fast statistical parsing with parallel multiple context-free grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, Gothenburg.

Becker, T., A. K. Joshi, and O. Rambow. 1991. Long-distance scrambling and Tree Adjoining Grammars. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 21–26, Berlin.

Blunsom, P. and T. Cohn. 2010. Unsupervised induction of tree substitution grammars for dependency parsing. In *Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1204–1213, Massachusetts.

Bochmann, G. V. 1976. Semantic evaluation from left to right. *Communications of the ACM*, 19(2):55–62.

Bodirsky, M., M. Kuhlmann, and M. Möhl. 2005. Well-nested drawings as models of syntactic structure. In *Proceedings of the 10th Conference on Formal Grammar and 9th Meeting on Mathematics of Language*, pages 195–203, Edinburgh.

Böhmová, A., J. Hajič, E. Hajičová, and B. Hladká. 2000. The Prague dependency treebank: A tree-level annotation scenario. In A. Abeillé, editor, *Treebanks: Building and Using Syntactically Annotated Corpora*, Kluwer, Dordrecht, pages 103–127.

Boyd, A. 2007. Discontinuity revisited: An improved conversion to context-free representations. In *Proceedings of the Linguistic Annotation Workshop, at ACL 2007*, pages 41–44, Prague.

Brainerd, W. S. 1969. Tree generating regular systems. *Information and Control*, 14:217–231.

Brants, S., S. Dipper, P. Eisenberg, S. Hansen-Schirra, E. König, W. Lezius, C. Rohrer, G. Smith, and H. Uszkoreit. 2004. TIGER: Linguistic interpretation of a German corpus. *Research on Language and Computation*, 2:597–620.

Bresnan, J., R. M. Kaplan, S. Peters, and A. Zaenen. 1982. Cross-serial dependencies in Dutch. *Linguistic Inquiry*, 13(4):613–635.

Buchholz, S. and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Natural Language Learning*, pages 149–164, New York.

Campbell, R. 2004. Using linguistic principles to recover empty categories. In *42nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 645–652, Barcelona.

Charniak, E. 1996. Tree-bank grammars. In *AAAI 96 Proceedings*, pages 1031–1036, Portland, OR.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *6th Applied Natural Language Processing Conference and 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139 (Section 2), Seattle, WA.

Charniak, E., K. Knight, and K. Yamada. 2003. Syntax-based language models for statistical machine translation. In *Proceedings of the Ninth Machine Translation Summit*, pages 40–46, New Orleans, LA.

Chen, D. and C. D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 740–750, Doha.

Chomsky, N. 1981. *Lectures on Government and Binding,* volume 9 of *Studies in Generative Grammar*. Foris Publications.

Collins, M. 1997. Three generative, lexicalised models for statistical parsing. In *35th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 16–23, Madrid.

Collins, M. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 175–182, Stanford, CA.

Courcelle, B. and P. Franchi-Zannettacci. 1982. Attribute grammars and recursive program schemes. *Theoretical Computer Science*, 17:163–191.

van Cranenburgh, A. 2012. Efficient parsing with linear context-free rewriting systems. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–470, Avignon.

Daniels, M. and W. D. Meurers. 2002. Improving the efficiency of parsing with discontinuous constituents. In *Proceedings of NLULP'02: The 7th International Workshop on Natural Language Understanding and Logic Programming*, pages 49–68, Copenhagen.

Daniels, M. W. and W. D. Meurers. 2004. A grammar formalism and parser for linearization-based HPSG. In *the 20th International Conference on Computational Linguistics*, volume 1, pages 169–175, Geneva.

Deransart, P., M. Jourdan, and B. Lorho. 1988. *Attribute Grammars: Definitions, Systems and Bibliography,* volume 323 of *Lecture Notes in Computer Science*. Springer-Verlag.

Deransart, P. and J. Małuszynski. 1985. Relating logic programs and attribute grammars. *Journal of Logic Programming*, 2:119–155.

Duske, J., R. Parchmann, M. Sedello, and J. Specht. 1977. IO-macrolanguages and attributed translations. *Information and Control*, 35:87–105.

Eisner, J. 1996. Three new probabilistic models for dependency parsing: An exploration. In *the 16th International Conference on*

Computational Linguistics, volume 1, pages 340–345, Copenhagen.
Engelfriet, J. and G. Filè. 1981. The formal power of one-visit attribute grammars. *Acta Informatica*, 16:275–302.
Engelfriet, J. and E. M. Schmidt. 1977. IO and OI. I. *Journal of Computer and System Sciences*, 15:328–353.
Engelfriet, J. and E. M. Schmidt. 1978. IO and OI. II. *Journal of Computer and System Sciences*, 16:67–99.
Engelfriet, J. and H. Vogler. 1998. The equivalence of bottom-up and top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 56:332–356.
Evang, K. and L. Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin.
Fischer, M. J. 1968. Grammars with macro-like productions. In *IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory*, pages 131–142, Schenectady, NY.
Fouvry, F. and D. Meurers. 2000. Towards a platform for linearization grammars. In *Proceedings of the Workshop on Linguistic Theory and Grammar Implementation, at ESSLLI-2000*, pages 153–168, Birmingham.
Fülöp, Z. and H. Vogler. 1999. A characterization of attributed tree transformations by a subclass of macro tree transducers. *Theoretical Computer Science*, 32:649–676.
Gabbard, R., S. Kulick, and M. Marcus. 2006. Fully parsing the Penn Treebank. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 184–191, New York.
Gebhardt, K. and J. Osterholzer. 2015. A direct link between tree-adjoining and context-free tree grammars. In *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP 2015)*, Düsseldorf.
Gécseg, F. and M. Steinby. 1997. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 3*. Springer, Berlin, chapter 1, pages 1–68.
Giegerich, R. 1988. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423.
Gómez-Rodríguez, C., M. Kuhlmann, and G. Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, pages 276–284, Los Angeles, CA.
Gómez-Rodríguez, C., M. Kuhlmann, G. Satta, and D. Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 539–547, Boulder, CO.
Gómez-Rodríguez, C. and G. Satta. 2009. An optimal-time binarization algorithm for linear context-free rewriting systems with fan-out two. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 985–993, Suntec.
Gorman, Kyle. 2016. Pynini: A Python library for weighted finite-state grammar compilation. In *Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata*, pages 75–80, Berlin.
Henderson, J. 2004. Discriminative training of a neural network statistical parser. In *42nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 95–102, Barcelona.
Hockenmaier, J. and M. Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.
Johnson, M. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *40th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 136–143, Philadelphia, PA.
Kahane, S., A. Nasr, and O. Rambow. 1998. Pseudo-projectivity, a polynomially parsable non-projective dependency grammar. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 646–652, Montreal.
Kallmeyer, K. and M. Kuhlmann. 2012. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Eleventh International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 108–116, Paris.
Kallmeyer, L. and W. Maier. 2010. Data-driven parsing with probabilistic linear context-free rewriting systems. In *the 23rd International Conference on Computational Linguistics*, pages 537–545, Beijing.

Kallmeyer, L. and W. Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.
Kallmeyer, Laura. 2010. *Parsing Beyond Context-Free Grammars*. Springer-Verlag.
Kanazawa, M. 2009. The pumping lemma for well-nested multiple context-free languages. In *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 312–325, Springer-Verlag, Stuttgart.
Kanazawa, M. and S. Salvati. 2010. The copying power of well-nested multiple context-free grammars. In *Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 344–355, Trier.
Kathol, A. and C. Pollard. 1995. Extraposition via complex domain formation. In *33rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 174–180, Cambridge, MA.
Kepser, S. and J. Rogers. 2011. The equivalence of tree adjoining grammars and monadic linear context-free tree grammars. *Journal of Logic, Language and Information*, 20:361–384.
Klein, D. and C. Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *42nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 478–485, Barcelona.
Klein, D. and C. D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 40–47, Edmonton.
Knuth, D. E. 1968. Semantics of context-free languages. *Mathematical Systems Theory*, 2:127–145. *Corrections in Mathematical Systems Theory*, 5 (1971):95–96.
Kübler, S., R. McDonald, and J. Nivre. 2009. *Dependency Parsing*, volume 2(1) of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers LLC.
Kuhlmann, M. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2):355–387.
Kuhlmann, M. and G. Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 478–486, Athens.
Kuhlmann, Marco and Joachim Niehren. 2008. Logics and automata for totally ordered trees. In *Rewriting Techniques and Applications: 19th International Conference*, pages 217–231, Hagenberg.
Lu, W., H. T. Ng, W. S. Lee, and L. S. Zettlemoyer. 2008. A generative model for parsing natural language to meaning representations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 783–792, Honolulu, HI.
Maier, W. and L. Kallmeyer. 2010. Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing. In *Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 119–126, New Haven, CT.
Maier, W. and A. Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proceedings of the 13th Conference on Formal Grammar*, pages 61–76, Hamburg.
Maier, Wolfgang and Timm Lichte. 2009. Characterizing discontinuity in constituent treebanks. In *Proceedings of the 14th Conference on Formal Grammar*, volume 5591 of *Lecture Notes in Artificial Intelligence*, pages 167–182, Bordeaux.
Maletti, A. and J. Engelfriet. 2012. Strong lexicalization of tree adjoining grammars. In *50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 506–515, Jeju Island.
Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.
McCawley, J. D. 1982. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1):91–106.
McDonald, R. and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, Trento.
McDonald, R., F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver.
Mel'čuk, I. A. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.
Mönnich, U. 2010. Well-nested tree languages and attributed tree transducers. In *Tenth*

International Workshop on Tree Adjoining Grammar and Related Formalisms, pages 35–44, New Haven, CT.

Müller, S. 2004. Continuous or discontinuous constituents? A comparison between syntactic analyses for constituent order and their processing systems. *Research on Language and Computation*, 2:209–257.

Nederhof, M.-J. and H. Vogler. 2014. Hybrid grammars for discontinuous parsing. In *the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1370–1381, Dublin.

Nivre, J. 2010. Statistical parsing. In N. Indurkhya and F. J. Damerau, editors, *Handbook of Natural Language Processing. Second Edition*. CRC Press, Taylor and Francis Group, pages 237–266.

Nivre, J. and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 99–106, Ann Arbor, MI.

Nivre, Joakim. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec.

Nivre, Joakim, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *LREC 2006: Fifth International Conference on Language Resources and Evaluation, Proceedings*, pages 2216–2219, Genoa.

Nivre, Joakim, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 73–76, Paris.

Paakki, J. 1995. Attribute grammar paradigms — a high level methodology in language implementation. *ACM Computing Surveys*, 27(2):196–255.

Petrov, S., L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney.

Pollard C. and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Rambow, O. 2010. The simple truth about dependency and phrase structure representations: An opinion piece. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, pages 337–340, Los Angeles, CA.

Rambow, O. and A. K. Joshi. 1997. A formal look at dependency grammars and phrase structure grammars with special consideration of word-order phenomena. In L. Wenner, editor, *Recent Trends in Meaning-Text Theory*. John Benjamin, pages 167–190.

Ranta, Aarne. 2011. *Grammatical Framework: Programming with Multilingual Grammars*, CSLI Publications, Stanford, CA.

Reape, M. 1989. A logical treatment of semi-free word order and bounded discontinuous constituency. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 103–110, Manchester.

Reape, M. 1994. Domain union and word order variation in German. In J. Nerbonne, K. Netter, and C. Pollard, editors, *German in Head-Driven Phrase Structure Grammar*. CSLI Publications, Stamford, CA, pages 151–197.

Rounds, W. C. 1970. Mappings and grammars on trees. *Mathematical Systems Theory*, 4:257–287.

Satta, G. and M. Kuhlmann. 2013. Efficient parsing for head-split dependency trees. *Transactions of the Association for Computational Linguistics*, 1:267–278.

Satta, G. and E. Peserico. 2005. Some computational complexity results for synchronous context-free grammars. In *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 803–810, Vancouver.

Seifert, S. and I. Fischer. 2004. Parsing string generating hypergraph grammars. In *Proceedings of the 2nd International Conference on Graph Transformations*, volume 3256 of *Lecture Notes in Computer Science*, pages 352–267, Springer-Verlag, Rome.

Seki, H. and Y. Kato. 2008. On the generative power of multiple context-free grammars and macro grammars. *IEICE Transactions on Information and Systems*, E91-D:209–221.

Seki, H., T. Matsumura, M. Fujii, and T. Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229.

Shieber, S. M. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343.

Shieber, S. M. and Y. Schabes. 1990. Synchronous tree-adjoining grammars. In *Papers Presented to the 13th International*

Conference on Computational Linguistics, volume 3, pages 253–258, Helsinki.
Sima'an, K., R. Bod, S. Krauwer, and R. Scha. 1994. Efficient disambiguation by means of stochastic tree substitution grammars. In *Proceedings of International Conference on New Methods in Language Processing*, pages 50–58, Manchester.
Skut, W., B. Krenn, T. Brants, and H. Uszkoreit. 1997. An annotation scheme for free word order languages. In *Fifth Conference on Applied Natural Language Processing*, pages 88–95, Washington, DC.
Stucky, S. 1987. Configurational variation in English. In G. J. Huck and A. E. Ojeda, editors, *Discontinuous Constituency,* volume 20 of *Syntax and Semantics*. Academic Press, pages 377–404.
Vijay-Shankar, K. and A. K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *23rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 82–93, Chicago, IL.
Vijay-Shanker, K., D. J. Weir, and A. K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 104–111, Stanford, CA.